Intelligent Task-Level Grasp Mapping for Robot Control

ROYAL INSTITUTE
OF TECHNOLOGY

JOSEP MARIA COMAS JORDÀ

Master's Thesis at NADA
Supervisor: Staffan Ekvall, CVAP/CAS
Examiner: Jan-Olof Eklundh, KTH
October 2006

A tothom qui ho ha fet possible.

Abstract

In the future, robots will enter our everyday lives to help us with various tasks. For a complete integration and cooperation with humans, these robots need to be able to acquire new skills. Sensor capabilities for navigation in real human environments and intelligent interaction with humans are some of the key challenges.

Learning by demonstration systems focus on the problem of human robot interaction, and let the human teach the robot by demonstrating the task using his own hands. In this thesis, we present a solution to a subproblem within the learning by demonstration field, namely human-robot grasp mapping. Robot grasping of objects in a home or office environment is challenging problem. Programming by demonstration systems, can give important skills for aiding the robot in the grasping task.

The thesis presents two techniques for human-robot grasp mapping, direct robot imitation from human demonstrator and intelligent grasp imitation. In intelligent grasp mapping, the robot takes the size and shape of the object into consideration, while for direct mapping, only the pose of the human hand is available.

These are evaluated in a simulated environment on several robot platforms. The results show that knowing the object shape and size for a grasping task improves the robot precision and performance.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
AGI	Agents for Gesture Interpretation
ANN	Artificial Neural Networks
CCH	Color Cooccurrence Histograms
DOF	Degree of Freedom
FANN	Fast Artificial Neural Networks
FSR	Field and Service Robots
GBP	Gesture-Based Programming
GWS	Grasp Wrench Space
HMD	Head Mounted Display
HMM	Hidden Markov Models
HO	Human Operator
HRI	Human-Robot Interaction
IEEE	Institute of Electrical and Electronics Engineers
IROS	Intelligent Robots and Systems
NASA	National Aeronautics and Space Administration
NoB	Nest of Birds
NN	Neural Networks
PbD	Programming by Demonstration
PDA	Personal Data Assistant, hand held computer
RSJ	the Robotics Society of Japan
SLAM	Simultaneous Localization and Mapping
SMM	Sensory-Motor Map
USB	Universal Serial Bus

VPT View-Point Transformation

2D Two Dimensional

3D Three Dimensional

Chapter 1

Introduction

The next generation of robots will be placed in our homes and help us in everyday lives. The number of possible tasks these robots may help us in is huge and we cannot expect the robots to arrive pre-programed for all these tasks. The end-users of robots and devices will be unexperienced in programming and not familiar with classical ways of programming. Classical robot task programming requires an experienced programmer and a lot of tedious work. Hence, a system which allows the user to demonstrate the task in a natural way by manipulating real objects is nowadays an important field of research.

Two main technological steps will take robots from factories to homes. First development of perception and navigation capabilities in an unstructured, changing environments. Second development of the capability of continuous communication with humans, rapid learning/adaptation to new work tasks. The first step has almost been taken. The rapid development of sensor technology (especially commercial laser scanners) and continuous increase in processing power, allows for heavy image processing and Simultaneous Localization and Mapping (SLAM) techniques. This made it possible to allow slowly moving robots to enter into the same areas as humans. However, since the difference between performance of animals is huge, it becomes clear that a lot of work remains to be done.

The second step is still far from being completed. Traditional industrial robots are mechanically capable of changing tools and performing different work tasks. Due to the nature of factory work, the time between reprogramming is relatively long and therefore interactive communication and continuous learning is not needed. The most sophisticated programming methods allows task design, testing and programming off-line with a simulation tool without any contact with the robot itself. Commercial mobile robots like vacuum cleaners and lawn mowers are limited to a single task by their mechanical construction. A multitask service robot needs both mechanical flexibility and high level of "intelligence" in order to carry out and learn several different tasks in continuous interaction with the operator. Instead of being a "multitool", the robot should be capable of using different kinds of tool designed for humans. These tasks depend on the environment the robot will operate in and cannot be entirely preprogrammed. It is therefore important to equip the robot with capabilities that allow it to learn new tasks or refine the existing ones either through direct interaction with the environment or through a teaching process with a user.

The main demand for these robots is almost complete human-machine interaction. Human expects machines with humanlike capabilities but this still difficult to be provide. The main bottlenecks are physical robotic constrains and intelligence, the existing physical constraints can be avoid by systems more intelligent and better human-robot interaction. The interaction should be natural for human and adapted at very limited learning capabilities in current robotic systems. The interaction should be optimized between these two limits, robot intelligence and natural human demonstration.

1.1 Motivation

It is clear that to date, no robot hand with the same capabilities and flexibility as the human hand have been developed. The technology in the robotic field is advancing fast, but it have still not been possible for a robot hand to perform the wide list of different grasps of the human hand.

For a human it is easy to recognize an object and directly grasp this object as have been done thousands of times before. Helped by our cerebellum coordinates and all of our muscles and movements which allows us to perform all the every day movements in a intuitive way, avoiding the need to think with our brain. For our robots even if they are able to recognize the object, they cannot grasp it unless they know how. They can learn by training grasping on a common object, but it would be even better if they were able to relate the shape of the object to another object, previously known, and then perform a similar grasp only changing the approach for the differences on size of the object.

A Programming-by-Demonstration framework is based on robot imitation and learning from a human teacher by observation, to allow users without programming education to easily instruct a robot. The rapid development of sensors is giving the robots the capabilities to interact with humans in everydays environments, and the state of the art in PbD gives robots the possibility to learn how humans perform different grasps. We still have a big problem. Robots are not able to perform the same movements and grasps as humans. Depending on the kinematics of the manipulator different mappings between human and robot spaces are needed, a non-linear grasp mapping from human hand to the robot hand is necessary.

1.2 Project Goals

Understanding and interpreting dynamic scenes and activities is a very challenging problem. Programming-by-Demonstration is a flexible framework that reduces the complexity of programming robot tasks. PbD allows end-users to demonstrate the tasks instead of writing code, providing a programming system useful for every end-users and is able to learn a wide list of new tasks.

The basic idea in the field of programming-by-demonstration is to develop a system able of learning tasks demonstrated by a human teacher, in natural environments and with real objects. The system is given the ability to understand the real situation and transform it into the limited robot world, using a mapping from human hand to robot hand.

The goal is to develop an artificial cognitive system acting in everyday environments capable of learning actions commonly induced by humans, a system capable of learning pick-and place tasks by manually demonstrating them.

The main purpose of this project is to develop a system capable of learning robot tasks from demonstration. The system should be capable of recognition and modeling human grasps in a natural way. In this project we focus our work on robotic hands. As mentioned their capabilities are still far from a real human hand. Hence, when a PbD system is faced, our robots must know the existing differences between the robot manipulator and human hand, and apply an intelligent system to cope with these differences. For all of these reasons an Intelligent Grasp Mapping System is an important goal in robotics field.

1.3 Outline

Currently, our world is changing faster than ever, new objects are introduced in our life every day, and as humans we have a huge capability of adaptation and learning, for example *what is this new object?*, *how to use this new object?* are solvable questions for ourself. Adaptation was an opposite word with the traditional idea about robotics. The traditional robots are preprogrammed and they know what to do when the task has been programed before, but if they find a new challenge, they are not able to solve a new task or new situation that they have not been programmed for. To introduce these new capabilities for robots an expert programmer and a lot of tedious work is required. A solution of this traditional problem in robotics field is Programming by Demonstration.

In chapter 2, new techniques for controlling and programming robots are studied as Programming by Demonstration, that give robots the capability to learn and adapt to new objects and environments, by interaction with humans. Teleoperation and Human-Robot Interaction are both techniques based on the cooperation between human and robots, in order to perform certain tasks that otherwise would be dangerous or impossible to do for a human. Also, natural interfaces are studied to enable human-robot communication in an easy way, or in a specific way for disabled persons.

In chapter 3, is explained the general process of PbD systems, possible solutions for PbD implementation and grasp mapping situation within complete PbD system. Shows some possible applications for grasp mapping system. Also, at the beginning of this chapter is given a grasp classification.

To arrive to the goal of this project we must demonstrate the possibility for robot hands to learn a new non-linear mapping towards different everyday objects. The implementation of this system is explained in chapter 4. The system implementation was divided in three main phases:

- The data acquisition phase that in our case is done through a data glove using a general sensor device called Nest of Birds.
- The modeling phase using ANNs which is the main contribution of this project. Using this method, strategies are studied and evaluated to find the needed mapping between human and robot hands.
- The final mapping result is obtained using simulation with Graspit! simulator, a versatile tool that allow us to see and to evaluate our mappings in

a easy way without need of using expensive robotics systems not available to us.

In chapter5 we test and evaluate several mappings for different objects and robot hands. Using the system we model the mapping between the human hand space and the more limited robot hand space. This work consists of two steps. In the first step, a linear mapping is done between human hand and robot hand. In this part transmission of movement is direct and does not always make sense when an object is to be grasped. We call it direct or linear mapping, and it is motivated to call this a *non-intelligent mapping*. In the second step, we improve our system to get a non-linear mapping where the mapping depends on the object to be grasped. It has the ability to adapt to the object shape. Thus, we call this an *intelligent mapping*. To know and understand the possible grasps for a human hand and the possible resulting grasp applied for different robot hands. This classification gives the own robotic system the capability understand to *how* humans grasp objects.

Results obtained with different mapping systems are shown in chapter 6. For different systems and with different robotic hands and objects. In this part we study real capabilities of grasp mapping systems.

Finally, in Chapter 7 we extract some conclusions of our work done and comment on possible future work, how the system could be improved and which new features could be added.

Chapter 2

Related Work

In the literature study of this thesis we have studied several publications in the field of Programming-by-Demonstration(PbD), as well as different techniques in robot grasping, the main purpose of this project. Also, Artificial Neural Networks(ANN) have been studied because they are used throughout the project to learn from human input. Below is a short introduction of the work previously done in PbD, and a more detailed description is provided in the next chapter.

Much of the research that has been done in PbD considers Intelligent Grasp Mapping to be an important technique in order to do real interactive grasp mapping between human and robot hands. For example, [3] discuss a system capable of learning and replicating tasks demonstrated by human teacher, and how this can be integrated to a PbD solution. In [5], the general process of a PbD system is described. It is based on *Demonstration, Action Segmentation, Interpretation, Abstraction, Mapping* and *Execution*. The abstraction design depends on the purpose of the system. In general, each task is decomposed into a number of actions. In these papers the interpretation system is built from two independent modules: Object Recognition and Grasp Recognition, concerned on understanding *what* has been demonstrated, rather than *how* to execute the task.

A common learning strategy is to generate a number of training sequences and build a representation that later is used for on-line control of the robot hand motion. One of the problems encountered in human based learning settings is how to record the human action. A popular approach here is to use sensors such as a data glove or a magnetic tracker to retrieve the hand and fingertip positions. In [6],[7] and [8] PbD systems using a data glove is used to learn mappings between a human hand and several robot hands.

In [9], a setup is presented for controlling a four-finger robot hand using a dataglove. They present a solution for mapping human and artificial hand workspace using a neural network to implement a nonlinear learning calibration, that enables the operator to intuitively and easily telemanipulate objects with the artificial hand.

Other techniques have been studied, e.g., [10], where two new techniques for Gesture Imitation are presented, Sensory-Motor Map (SMM) and View-Point Transformation (VPT). The VPT allows the robot to map observed gestures to canonical point-of-view. The final step consists in transforming these mapped features to motor commands, which is referred to as the SMM. The SMM can

be computed explicitly if the parameters of the arm-hand-eye configuration are known a priori, and it can also be learned from observations of arm/hand motions.

Gesture-Based Programming(GBP) is a form of programming by human demonstration. The technique relies on the existence of previously acquired robotic skills, which are approximately the robotic equivalents of everyday human experiences. The interpretation of the human demonstration and matching to robotic primitives is a qualitative problem. In [11], they made an approach for this problem using skilled agents. The defined Agents for Gesture Interpretation are *Gesture Recognition Agents* and *Gesture Interpretation Agents*. They are interested in the interpretation of four types of manual and articulatory gestures for GBP. These include *symbolic gestures*, such as the pre-shape of dexterous grasp, *motion gestures*, such as transporting or manipulating an object, *tactile gestures*, such as fingertip contact force, and *articulatory gestures*, which are utterances such as when an error is made.

Not only PbD techniques have been studied, because despite significant advances in the field of artificial intelligence, human intelligence is far superior in terms of reasoning, language comprehension, vision, and ingenuity, among others. Some tasks may require both the acute reasoning and perceptive abilities of a human, and the strength and precision of a mobile manipulator. Therefore, it is good to design a control scheme such that humans may easily cooperate with the robots. Hence, another type of robot control system, teleoperation, was studied in [12],[1],[13]. A teleoperation system allow people to control machines in remote locations, generally inaccessible to humans. Because of its many benefits, such as increasing the reachability and safety of humans, the topic of teleoperation has been studied extensively for decades[14]. Controlling mobile robots through teleoperation is a challenging task that demands a flexible and efficient user interface. Mobile robots are often equipped with numerous sensors(proximity sensors, system status sensors, positioning and heading devices, multiple cameras, etc.) that provide a high volume of data to the user[15]. The area of teleoperation systems will be explored more thoroughly in the next section.

Related to PbD and teleoperation systems, Human-Robot interaction is a key subject that let robots enter in everyday human life. Human-Robot interaction and Natural Interfaces show us that, while controlling a service robot, the operator has several possibilities when interacting with the robot.

2.1 Teleoperation

Robots are slowly moving from factories to mines, construction sites, public places and homes. This new type of robots should be capable of performing different kinds of tasks in unstructured changing environments, not only among humans but through continuous interaction with humans. The main requirements for service robots(FSR) are mobility, advanced perception capabilities, high "intelligence" and easy interaction with humans. Although mobility and perception capabilities are no longer bottlenecks, they can nevertheless still be greatly improved. The main bottlenecks are intelligence and the human-robot interaction(HRI). Despite huge efforts in "artificial intelligence" research, the robots and computers are still very "stupid" and there are no major advance-

ments on the horizon. This emphasizes the importance of HRI. In the subtasks, where high-level cognition or intelligence is needed, the robot has to ask for help from the operator. In addition to task commands and supervision, the HRI has to provide the possibility of exchanging information about the task and environment through continuous dialogue. Furthermore, methods for direct teleoperation, and robot interfaces based on high-level cognitive interaction, are needed.

Unhanded vehicles have developed from teleoperated machines to multitask service robots during the last 60 years. Some applications are in remote unstructed environments such as battlefield robotics, (nuclear) waste cleanup, construction in space, underwater operations, etc. Such tasks are not foreseen to be achievable with high degrees of autonomy, due to cost constraints, technological unfeasibility, and/or high levels of operational uncertainty, and will consequently require mediation by a human operator(HO). In general, the role of the HO in teleoperation systems is expected to range anywhere from continuous close loop manual control to monitoring semi-automated systems with some degree of local control autonomy.

Mechanical manipulators developed rapidly to electric servos and finally to unmanned vehicles. Vehicle teleoperation was enabled by the possibility of transferring control and image data between the vehicle and the operator. This can already be seen as a primitive HRI. Despite the fact that a teleoperated vehicle does not fulfill any of the definitions of a robot, it can still be seen as a precursor to the robot. Step-by-step, the teleoperation technology was improved to telepresence-based technology by increasing the sensory feedback and the possibility of the operator controlling the sensor positions. Both traditional teleoperation and telepresence include the operator as "a main processor" in the control loop[16]. All the sensors information is brought to the operator and he controls all the actuators directly. In telepresence, the amount of sensor data and controllable actuators have been increased with the help of special interface devices like Head Mounted Display HMDs, head trackers, datagloves, etc.[17]

Robotics emerged along with digital processor technology, which made it possible to program autonomous functions of a vehicle or a manipulator. Most of today's robots are manipulators, which typically repeat the same task continuously and are reprogrammed infrequently. Typically, the operator's help is not needed during the operation. The existing commercial service robots, like vacuum cleaners and lawn mowers, are similar types of single-task robots. The operator simply switches them on and leaves them to perform the task. If the robot has problems during the task, e.g., gets stuck, it has to wait until the operator notices the situation and solves the problem.

The situation changes dramatically when the robot has to solve several more complicated tasks, including object/environment recognition and manipulation. Despite the continuously increasing computing power and development in "artificial intelligence", the autonomous abilities of robots remain very limited. In all the more complicated tasks, human help is needed. Human help can take the form of direct teleoperation or higher-level action like giving verbal instructions, but in all cases it involves the human as part of the task execution control loop of the robot[18]. The human effort can be either pre-planned or non-planned. The non-planned effort is usually called intervention[19],[18] and it can be activated by both the robot and the operator. However the need for the operator effort decreases the autonomy of the robot, and increases both the operator load

and the transferred amount of information between the robot and the operator. The development of the needed operator effort between a robot and an operator is illustrated in Figure 2.1 .

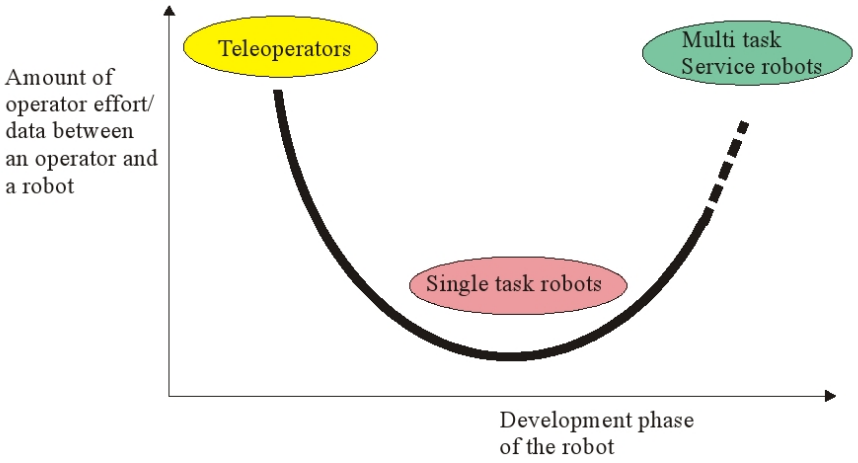

Figure 2.1: Development of operator effort in robotics, source [1].

Instead of a single robot, the robot can also be of a type that includes several independent entities[20]. One operator is not capable of operating such a swarm by direct teleoperation. However, increasing the autonomy of the robots and operator actions to task level, the teleoperation of swarming robots is also possible and can be achieved based on the same principles as the control of a single robot.

The aim of the robots is to serve and help humans, not vice versa. In a way, the single-task is perfect: human intervention is not needed during the task and the operator is released for other tasks. If the human effort is anyway needed to control the robot, it has to be in a sensible relation to the realized payback of the robot performance in a certain task. As mentioned before, the formula is totally task dependent. In some simple tasks, even a minimal intervention of the operator is too much, whereas the unmanned aerial vehicle Predator requires continuous control of several operators[21]. The best way to study this optimization problem is to use the well-known definition of usability[22]. First of all, *effectiveness:* Can the robot complete the commanded tasks and achieve the goals? If yes, the next question relates to *efficiency:* How much effort does the robot, and especially the operator, require to perform the task? If the operator effort is too big in comparison to the result, the robot should not be used at all. And finally comes *satisfaction:* What does the operator think about the ease of use?

2.1.1 Definition: Teleoperation

The term *teleoperation* refers simply to the operation of a vehicle or system over a distance [23]. Broadly, all interaction with a mobile robot comes under this definition. Traditionally, teleoperation is divided into direct teleoperation and supervisory control. In direct teleoperation, the operator closes all control loops

himself, while when in supervisory control a remarkable amount of the control is exercised by the teleoperator, i.e., the teleoperator is a robot. The term *teleoperation* refers to direct teleoperation, while supervisory control is handled under human-robot interaction. In today's digital world it has to be noted that even in the case of the direct teleoperation there usually exist control loops in the teleoperator. Typically these loops control the position or the velocity of the "directly" controlled actuators. The author has used the term *coordinated teleoperation* in [24] to separate this case from supervisory control. Here the teleoperator is always a moving work-machine/robot. In the case of a stationary robot the situation is slightly different. If the human is in the vicinity of the robot, he perceives more information than just the robot sensor values. Hence, the control of the robot is more simple.

Teleoperation information from [1] and [24].

2.2 Human-Robot Interaction

As mentioned in the introduction of this chapter, the service robots need human help as part of the task-processing loop. The help can be occasional advice[18], periodic teleoperation [25], continuous dialogue [26] or some combination of these. The most intensive operator effort is needed when new tasks are taught to the robot. The main interest is usability, i.e., how efficiently the operator can perform a task with the robot and how satisfied she/he is in the work process. To reduce the operator effort, it is important to evaluate the usability of possible interfaces in human-robot interaction.

While controlling a service robot, the operator has several possibilities open to him when interacting with the robot. Commands can be given traditionally via a computer interface [27], by speech [28], by gestures [27] or even by brain waves [29] and the robot can be directly controlled by means of different kinds of teleoperation devices. Fong [23] has divided the interfaces to four groups: *direct, multimodal/multisensor, supervisory control* and *novel*, according to the functional principle, except in the case of the last one, which relates to the relative novelty of the interface causing the possibility of misconceptions. The other problem is the word *supervisory*, which covers a very large area and causes a lot of overlapping with the others. Despite the shortcomings, the presented classification is quite clear. It is definitely difficult to classify the interfaces into closed groups. Here the interfaces are presented and classified according to the primary area of interaction they are used in. The starting point is that a service robot interface is always supervisory, but consists of several subinterfaces, which are classified in following way:

- **Command and dialogue interfaces**
  Interfaces to give commands and general information to the robot and receive the robot reply, questions and state information. Typical examples: computer/PDA interfaces, speech and gesture interfaces.

- **Direct control interfaces**
  Closed loop control interfaces to control movements of the robot or its manipulators. Typical examples: joysticks, manipulator controllers and telepresence equipment.

- **Spatial information interfaces**
  Interfaces to help the robot and operator to understand the environment and to fix locations and objects in common coordinates. Typical examples: map interfaces, pointers and cameras.

The command and dialogue type of interface is always needed for a robot. To fulfill its definition, a robot should be controllable. At its simplest, the commanding interface is just an on/off button, as in vacuum cleaning robots.

2.3 Natural Interfaces

As the name indicates, these should be as natural as possible for human to use. To fulfill this requirement, interfaces should mimic human communication as much as possible. The main method for human communication is speech, but in face-to-face communication, gestures and expressions have a lot of significance, sometimes much more than we think [30].

Speech is the most obvious way to transfer information to a service robot. Commercial speech (voice) controlled interfaces are already available in mobile phones and computers. In the case of mobile phones, speech is used just for giving the calling command and the name (e.g., "call Ruben"). New speech processing software like Philips[31] provides the possibility of controlling the computer by speech and even of recognizing dictation. There have also been a lot of experiments in controlling robots by speech. However, recognition seems still to be a problem, especially in environments with changing background noise.

Gestures and expressions are a very important part of human communication. Usually they are used in addition to speech to intensify the message, but sometimes they can be the only method of communication. Typical examples of gesture communication are the sign languages of deaf people, the maritime flag signaling, the referee's signs in different sport and different gestures used in military and vehicle control. In robotics, gestures make communication possible, especially in noisy environments, and provides a possibility for pointing which makes it possible to provide additional information, along with other communication methods. Dynamic gestures can also be easily recognized by inertial sensors, located in hand or in handheld devices like mobile phones or PDAs. "Primate" or humanoid-type robots can also use their manipulators to reply with gestures.

These natural interfaces can be helped by the direct interfaces used in closed loop teleoperation of a robot or its subsystem. Traditional examples are joysticks, driving wheels, pedals, mechanical hand/finger trackers like data gloves etc. However, the hardware of these interfaces is almost fully developed, and thus the possibility of new contributions relies in the software. The new innovations are related to gaze control and different bio controllers like myoelectric control [32] and brainwave control [29].

Chapter 3

Background

One of the ways of programming robots in a natural and easy way is to allow the user to first demonstrate the task and equip the robot with the functionality to learn from observation. An artificial system can retrieve a large amount of knowledge, simply by looking at other individuals, humans or robots working in the same area. In fact, similarly to human infants, a robot could learn significant information if it were able to recognize and imitate what others are doing. Nowadays, the demand for flexible and re-programmable robots has increased the need for programming-by-demonstration systems.

Learning-by-Demonstration frameworks, where the robot observes the human performing a task and is afterwards able to perform the task itself, use different sensory modalities for observation: vision, force-torque and haptics. To make the learning problem tractable, a hierarchical representation is commonly used depending on the complexity of the task at hand. Systems in which users demonstrate their task to the system, and then the system learns a program that accomplishes the task is a goal of PbD. The benefits of PbD are that the user does not need to learn an arcane programming language in order to automate repetitive tasks.

PbD is a remarkable method that allows end users to create, customize, and extend programs by demonstrating what the program should do. Until recently most programming power has been in the hands of the professional programmer rather than the end user. PbD is a method that allows end users by demonstration improve the old programs or teach new programs without knowledge of programming language. These techniques should allow the user with minimal programming expertise to create highly interactive software. Researchers have applied PbD techniques in order to reduce the amount of programming required for interactive software. A PbD system uses inductive learning to generate a program from example demonstrations.

To produce a program, the user draws the various interface components and then demonstrates their behavior by mimicking the desired program response to various events. The system records and analyzes the demonstrations, converting the low-level demonstrated actions into a higher-level description of the program behavior.

PbD allows learning complex functions from examples of their input and output behavior. The purpose is to have the user demonstrate the result of a computation, from which the system is able to construct a generalized program

which is able to accomplish the user's task.

Grasp recognition is considered to be part of a PbD framework. The user demonstrates the task by manipulating objects in a natural way. The objects are arranged to the desired configuration, and the system then automatically builds the task description. The basic idea is develop a system able of learning tasks demonstrated by a human teacher, the user demonstrates the task, and various sensors such as cameras observe and store the demonstration. The user notifies the system when the demonstration starts and when it is complete. Understanding and interpreting these dynamic scenes and activities is the authentic challenge.

3.1 Programming by Demonstration

Let us assume that we want to develop a PbD system able of learning tasks demonstrated by humans in natural environments, like everyday object grasping. As we see in [3], [33], this process can be summarized in different phases. From the user demonstration of a specific task which is observed for the robot by a sensoring system, to the final state where the robot is able to replicate this task by itself. To show this process more clearly, we have divided the complex process of learning a task in different phases from the robotic system's point of view, see Figure 3.1 . The PbD process can be divided into four phases:

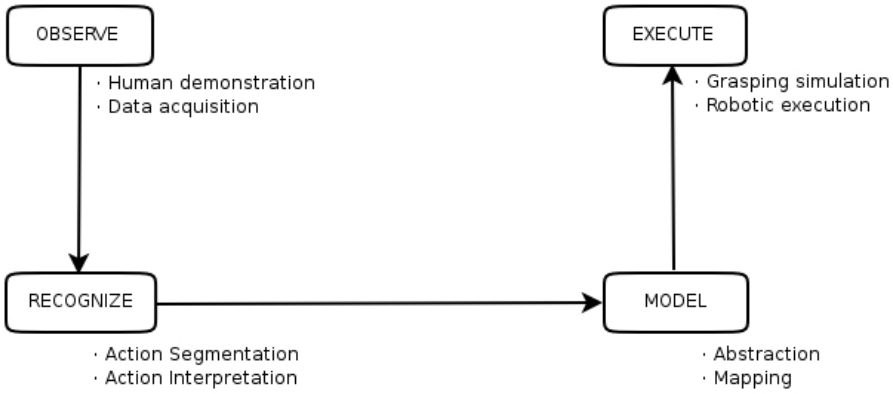

Figure 3.1: The general process of robot Programming by Demonstration.

1. **Observation Phase.**
   The robot observes the user demonstration. The sensor system is used for observing the user movements and actions, and important changes in the environment like object positions and task constraints. Various sensors are used for this purpose, such as cameras, magnetic trackers and data gloves.

2. **Recognition Phase.**
   This phase consists of two main processes. First, action segmentation which is the filtering of irrelevant sensor data, and identification of subtasks which are segmented. Next, the action interpreter transforms the

sensor data into meaningful actions, here the system extracts what the demonstrator is doing.

3. **Modeling Phase.**
   Here the interpretation of each subtask is stored as a task knowledge in a form that it is reuseable even if execution conditions changes slightly from the demonstration conditions. The entire demonstration is abstracted into a set of linked key identifiers. This is the abstraction of the current demonstration. Also in this phase a mapping is done from the abstraction symbols types, and trajectories are calculated for the robotic platform. For this, additional background knowledge about the kinematic structure of the target system is required.

4. **Execution Phase.**
   The execution can be done by simulation or by executing the task on a real robotic platform. In simulation, the system is tested in the simulated environment. This allows the user to confirm the correctness while avoiding dangerous situations in the execution case. For a real execution, success and failure can serve as feedback for modifications on the current mapping strategy.

Learning a Known Task

To simplify the complex process of learning new tasks another strategy is studied, see Figure 3.2 . This is a training of a specific task by human demonstration, where the task is known to the robot, so the robot knows what the human demonstrator is doing and it does not have to recognize every parameter of the demonstrated scene. In this process the human demonstrator has direct contact with the robot, providing it with the missing parameters for full interpretation of a scene. This type of teaching can help robots when faced with a new PbD problem.

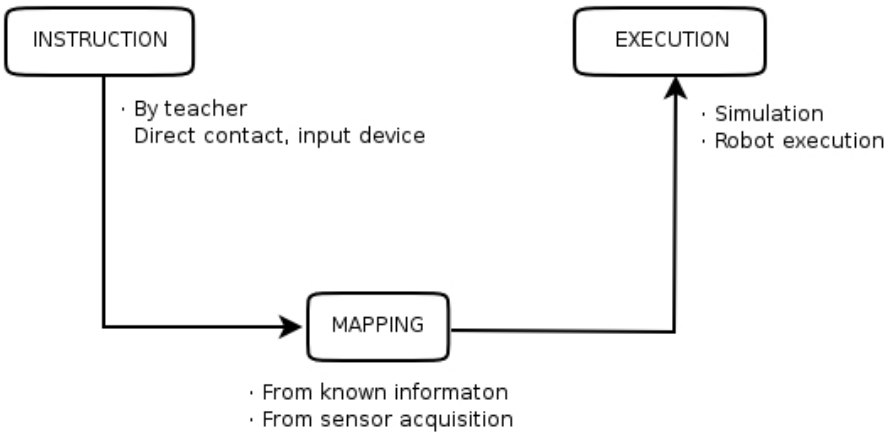

Figure 3.2: The Programming by Demonstration process when the task is known to the robot.

The main differences and advantages from the general process of PbD are listed below.

- The robot observes the user task demonstration and stores this information together with the task knowledge. That gives the robotic platform the necessary skills to easily repeat similar actions, when encountered with a similar known situation.
- More direct teaching can be done, for example the human can help the robot with an input device to perform the known task. This device can be a joystick or a similar robotic controller. Even direct contact methods can be used, by taking the robotic device, i.e. hand or arm, and performing the movements using force/torque sensors.
- The interpretation and abstraction are done by the human demonstrator, avoiding possible robotic errors and imprecision.
- This system gives more useful information than a simple robot observation but also requires more human implication and some experience in robotics from the human teacher.

3.1.1 Implementation of PbD System

Figure 3.3 on Page 15 shows a complete PbD system. It shows all the steps from human demonstration to robotic execution. As seen, this is a complex process. The focus in this thesis is on the implementation of an intelligent grasp mapping system. In order to understand how a PbD system works, the different components are briefly explained in the following sections, with examples of possible solutions for a future complete PbD system.

- The observation is done using sensors. There are numerous different robotic sensors, and they are briefly described in the next section. A more thorough description of the sensors relevant for this thesis is given in the next chapter.
- The object recognition is a key part of a PbD system. Some of the different strategies and systems done so far are described in section 3.4.2. These systems have been chosen since we believe they are easily integrated with our system.
- The modeling between human hand and different robotic hands is the goal of this project. Our implementation allows us to study and test different model configurations and our grasp mapping strategy.
- The execution on the robotic platform is the definite test of a PbD system, however this lies outside the scope of this work since we only implemented a part of a PbD system. However, in the result chapter are given experiments of the isolated mapping problem obtained by using robotic simulation. Also, in section 3.5 we provide some example applications for this system.

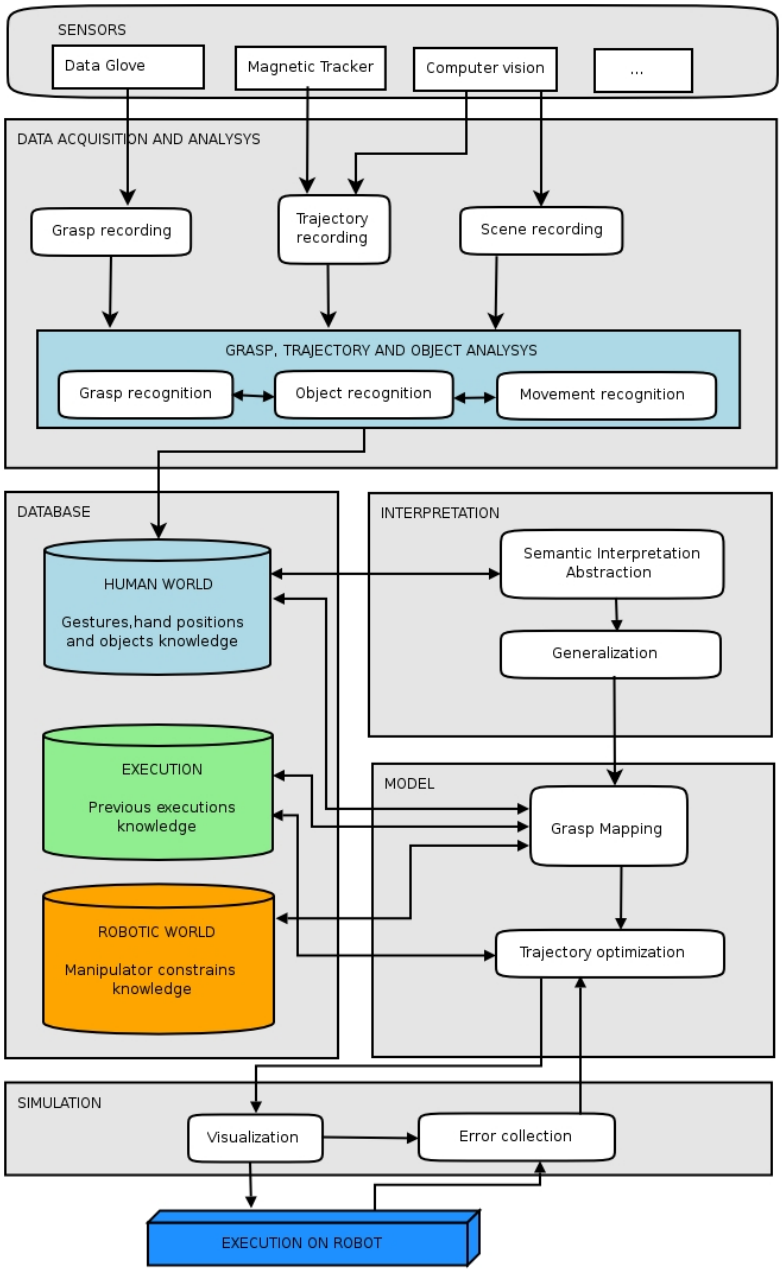

Figure 3.3: System structure for a Programming by Demonstration implementation.

3.2 Sensors

Let us assume robotic mobile manipulation in unstructured environments as a goal for a PbD implementation. This requires integration of a number of key research areas such as localization, navigation, object recognition, visual tracking/serving, grasping and object manipulation. It has been demonstrated that, given the above, and through simple sequencing of basic skills, a robust system can be designed [34]. In order to provide the robustness and flexibility required of the overall robotic system in unstructured and dynamic everyday environments, it is important to consider a wide range of individual skills using different sensory modalities, some of them introduced in this section. The sensory modalities can be classified in the following categories:

- Range
- Imaging
- Proximity
- Orientation
- Contact
- Internal
- Other

The laser scanner gives information about distance from objects. It is an example of range and proximity sensors, which are commonly used in SLAM applications rather than for activity recognition. Pressure sensors, measures the pressure distribution during a contact, is a classical contact sensor used in grasping field. Imaging sensors like cameras can give huge quantities of information, and this sensory system will be studied in following section. In our work we have used a magnetic tracking data glove, this sensor type are described in detail in the implementation chapter.

3.2.1 Computer Vision

Nowadays, cameras are considered key sensors for robots. Computer vision allows the robot to obtain wide information of the current environment, using various cameras and processing systems. The purpose of computer vision is to obtain as much as information as possible from images.

A drawback compared to the rest of the sensors is that cameras are are sensitive to changes in illumination and in general to the environment in which the action is learned. On the other hand, they do not have the problem of metal influence on sensor readings, like magnetic sensors have.

An alternative to the use of a data glove is to use a camera and computer vision to track the 3D pose and trajectory of the hand, at the cost of tactile feedback. But it has the disadvantages as mentioned before, sensitivity of environment, object occlusion etc. For all of these reasons we think that the best way is use a combination of these systems. Camera sensors can be used either as a complementary system of our dataglove or as a completely independent system. However, in the limited study of this thesis we only used a data glove.

3.3 Grasp Mapping

As the kinematics and configuration spaces of a human hand and an artificial robotic hand are generally different, the fingertip positions of the robotic hand cannot correspond exactly to the fingertip positions of the human hand (especially when fingertip grasps are considered). Grasp mapping is required to translate the recognized human hand posture, which is acquired from a data glove input and sensor devices, to the different robot hand available in the current simulated setup. A good mapping between the human and the artificial fingers positions is required. In general there are two ways of representing mapping between these two spaces, using:

- **Joint Space**
  Mapping using the joint space representation facilitates the *similarity* between hands poses. This is suitable for enveloping or power grasps.
- **Cartesian Space**
  Mapping using the Cartesian space is more suitable for representation of the fingertip positions. This is a natural approach when, for example, precision grasps are considered.

A third group may be added to this list, namely the combination of above. Here the positions of the human hand fingertips are mapped to some joint values of the robot hand. If the robot and the human have similar hand configurations, the result is likely to be the same as for purely Cartesian mapping. However, if the robot hand's kinematics is very different from the human one, as for example in the Barrett hand case, Cartesian mapping is not situable. The limitations and possible solutions for different robot hands will be studied in following chapters.

We are interested in learning the *grasp mapping* between a human hand and different types of robotic hands. This is done by interacting with common objects of everyday life. A simple data glove device is used as an input for human hand postures.

In following sections we show how a camera system used for the recognition of objects would be used in our system to to determine position and shape/size of the objects. The user's hand posture and the known shape and size of the object, is the input for our grasp mapping system. Hence, the system determines what is the best robotic hand posture for the given grasp, and how this mapping should be defined to use the full dexterity of the robotic hand.

Figure 3.4 on Page 18 show our grasp mapping strategy. This strategy is based on two main sources of information, the previous knowledge about the hand kinematics and shapes of everyday objects, and the grasp mapping training data. Both are important and complementary in the grasp mapping decision.

In order to do intelligent grasp mapping, the system needs knowledge about the human and robot world. We classify this *a priori* knowledge in three different categories: *a priori* knowledge of the human hand, *a priori* knowledge of manipulator, and *a priori* knowledge of common objects in the everyday world. This gives us the grasp constrains for robotic manipulator, and also helps to know *what* and *how* human is performing a task. With this information is possible to do grasp/object recognition, which provides key information for the robot grasp strategy.

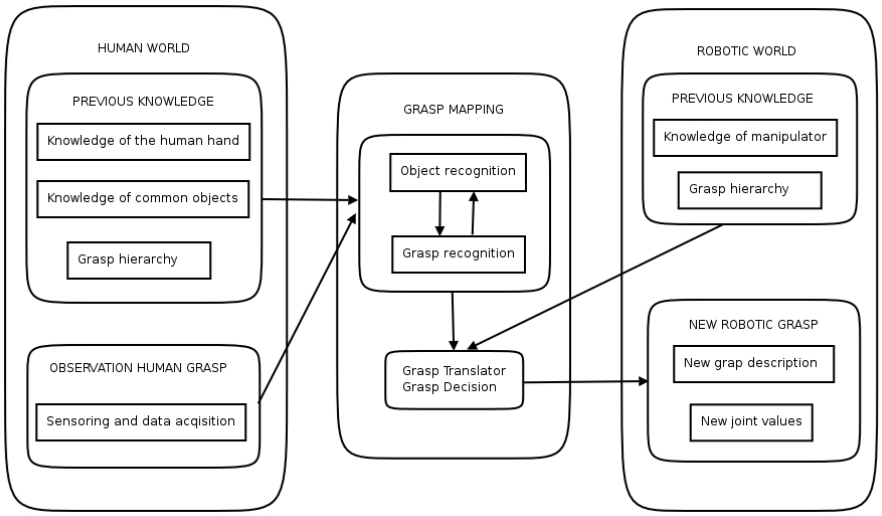

Figure 3.4: Grasp mapping strategy

The hierarchical description of different grasps is extracted from Cutkosky's work[35]. He summarizes a number of analytic measures, several of which are for the purpose of grasp synthesis. This can be seen in more detail in the next section, *Classifying grasps taxonomies*. Examples of such analytic measures are manipulability, grasp isotropy, force and form closure, internal forces and resistance to slipping.

3.3.1 Classifying Grasps Taxonomies

In order to classify grasps and represent different ways of grasping objects we base our modeling on Cutkosky's grasp hierarchy introduced in [35]. The Cutkosky's grasp taxonomy seen in Figure 3.5 on Page 19 is a grasp hierarchy which offers classification schema for typical grasps occurring doing everyday grasps and another common actions for a human hand. Hence, Cutkosky's grasp taxonomy have been used for the definition of different grasp classes. In this hierarchy, grasps are sorted into 16 different classes. Each class is either a power-grasp or precision-grasp, circular or prismatic. The basic idea is to perform the classification based on hand posture data. Then, the classification helps the system finding the most accurate robot grasp. In this work, all 16 grasps are not considered, and for a single robot grasp there may be several correspondant human grasps.

3.4 Dinamic scene interpretation

Interpretation is a key part of a Programming by Demonstration system. An interpretation system is usually built from two different modules, *Object recognition* and *Grasp recognition*. This approach allows the robot to know what the human demonstrator is doing and how he/she is performing the demonstration.

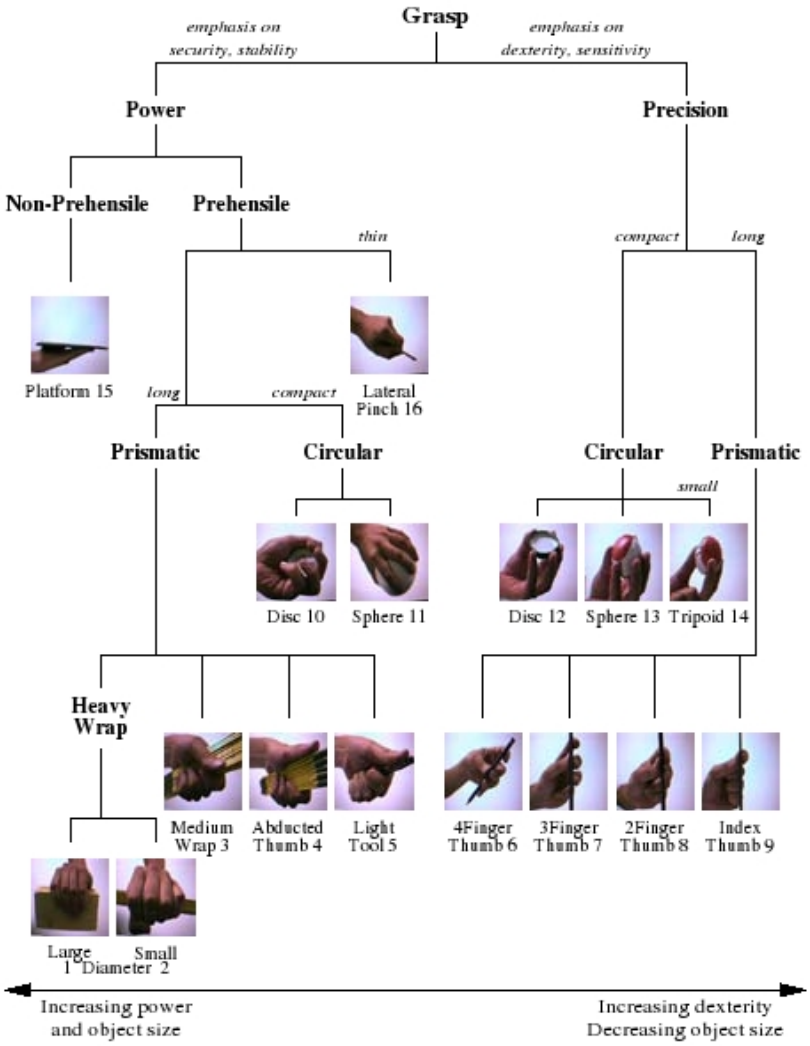

Figure 3.5: Cutkosky's grasp hierarchy.

The object class and location are estimated by the object recognition module, while the grasp used and trajectory movement or manipulation is estimated by the grasp recognition module. These two modules are closely related and the information of each one helps the other.

3.4.1 Grasp Recognition

The grasp recognition and grasp decision is based in the data extracted from the human demonstration, but previous knowledge is also important in this decision. The system has to know the real world constrains. The two following facts must be considered in grasp recognition task: i) A grasp depends on the target object, and ii) A grasp depends on the current task. For example: if the task is to take a cup to fill it with coffee, the object shape provides three possible grasps: a circular sphere grasp, a power wrap grasp around cup, and a two-finger-thumb precision grasp as seen in Figure 3.6 .

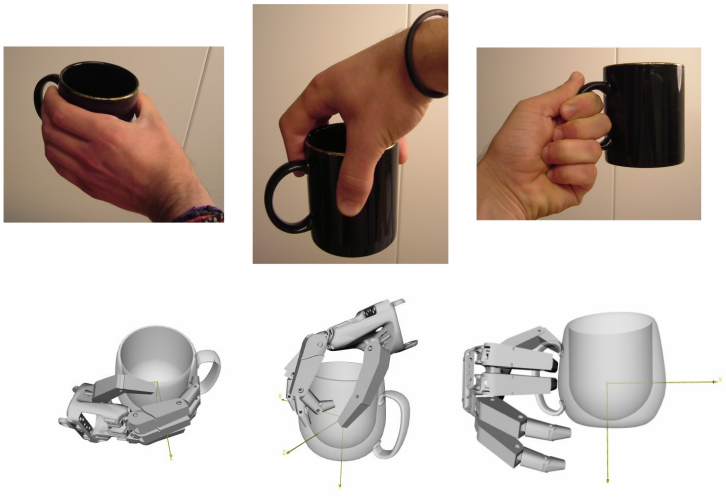

Figure 3.6: Different possible grasps for picking up a cup.

But the known task gives new limitations because in order to fill the cup the hole cannot be blocked. If the system knows the object and the task the group of possible grasps is reduced into only two, which makes the choice of grasp an easier decision. This reflection brings us to the top-down and bottom-up design.

Top-down and Bottom-up Design

- In the *top-down* model an overview of the system is formulated, without going into detail for any part of it. Each part of the system is then refined by designing it in more detail. Each new part may then be refined again, defining it in yet more detail until the entire specification is detailed enough to validate the model. Using the top-down model, a complete system can be designed without knowing the implementation details of every part.

- In contrast, in a *bottom-up* design individual parts of the system are specified in detail. The parts are then linked together to form larger components, which are in turn linked until a complete system is formed. Strategies based on this bottom-up information flow seem potentially necessary and sufficient because they are based on the knowledge of all variables that may affect the elements of the system.

For the grasp recognition in [3], Hidden Markov Models(HMM) are used to model the hand posture transitions during a grasp. The fingertip positions is captured by magnetic tracker, which requires the user to wear a dataglove. In [6] another method is presented, grasp classification based on arm movement trajectories. Here, the importance of the arm trajectory in the grasping process is evaluated. The magnetic tracker used in the HMM model is also used for trajectory recording, giving the position and orientation parameters of the hand during the grasping process. A hybrid method for dynamic classification is presented in [6], which combine HHM classification and hand trajectory classification. Ten grasps were considered from Cutkosky's taxonomy and the results showed a recognition ability of about 70% for a multiple user setting.

In [36] and [37] static hand posture classification has been investigated using Artificial Neural Networks. Here we give brief overview of these systems.

In [37], they implemented a method for grasp recognition in Master-Slave system. The 22 joint values of the human hand are captured using a data glove. The slave hand is rotationally symmetric and has nine DOFs. They designed a ANN grasp recognition and mapping, because is not possible do joint-to-joint mapping. The data glove recordings are written in a vector $V = [s_1, s_2, ...s_{22}]$. This vector is reduced to eight data channels to represent a hand posture $V_R = [s_1, s_2, ...s_8]$. A set of hand posture vectors $D_R = [V_{R1}, V_{R2}, ...V_{Rn}]$ is taken. This set is used to train the ANN. The ANN has eight inputs which is the length of the V_R vector. The output of the ANN is a vector of length three which is the number of grasps that the network should discriminate between. To train the neural network a set of 200 master hand postures vectors was taken for each grasp. Outputs of the ANN were used to do the recognition of three different grasps A, B and C.

In [36], they use Cutkosky's taxonomy [35] as a basis. They make a division of the grasp hierarchy into 10 different layers. As sensor system a 22 sensor data-glove was used. It gives 2 angular values for the wrist joint and 4 angular values for each finger representing the flexion of the three finger joints and one abduction angle. The 20 finger flexion and abduction values are pre-processed and used as input of the grasp classification process since these describe the posture of the user's hand. This classification obtained an overall result of 91.7% for a single-user, and 81.4% and 88.9% depending of ANN type for a multi-user. A similar approach has been adopted in [38].

3.4.2 Object Recognition

There are numerous different methods for object recognition. In our studies we have focused on the methods for object recognition from [3] and [2]. Objects are recognizable by an appearance based method, using Color Cooccurrence Histograms(CCHs). The CCH is an extension of regular color histograms. The CCH is an effective way to represent objects for recognition in images by keeping

track of pairs of pixels, it allows a variable amount of geometry to be added to the regular color histogram. CCH improves the recognition capabilities by capturing more of the texture features.

1. Color Cooccurrence Histograms: A CCH keeps track the of number of pairs of certain colored pixels that occur at certain separation distances in image space. By changing these distances, the sensitivity of the algorithm can be adjusted to geometric changes in the searched object appearance, caused by viewpoint change or object flexing. CCH is also robust to partial occlusions or cluttered backgrounds. see [39],[2].

2. Object Recognition using Color Cooccurrence Histograms: The system stores several model images for an object changing the viewpoint. The system computes each of these images using CCHs. When the new input image contains many objects, the searched object can not be found directly using CCHs. To find the object in a new image, the system make overlapping subimages from the input image and compute the CCH for each subimage. Then, by comparison of these to the CCH object model image, the most probable subwindow is identified. This process can be repeated for each object that system is searching for. In Figure 3.7(a) on Page 23 is shown a typical result for this technique to find an object.

3. Object Localization: Once the object has been found and recognized in an input image, its position can be determined. The object center is calculated from the vote matrix in two steps. First, a rough segmentation of the object is performed, by starting from the strongest vote cell, and gradually expanding the borders. The process is described in detail in [3]. Then, the position is calculated as the average positions of all vote cells in this segmentation. The position is returned in screen coordinates of the image by the object recognizer and then transformed into world coordinates.

3.4.3 Object Movement

Object movement can be detected by comparing the searched objects from different images, just taking frames of a video like sequence of images and doing the comparison of their positions. For location the above technique can be used. However, the CCH based recognition and location in a new image consumes a lot of time. To avoid this inconvenience and speed up the process [3] presents a solution for this problem. They used a fixed mounted camera, and only care about the temporal changes in the image, based on the known background because the background image is stored at the beginning of the demonstration. Then, during task demonstration, each pixel is compared with the background image, and only pixels which have changed color more than a certain threshold are kept. Naturally, this means that some parts of the moving object will also be removed, as illustrated in Figure 3.7(b) . Shadows is another issue, they may also reveal background, as seen in Figure 3.7(b) . Despite these problems we believe that this method could performs satisfactory for a PbD system.

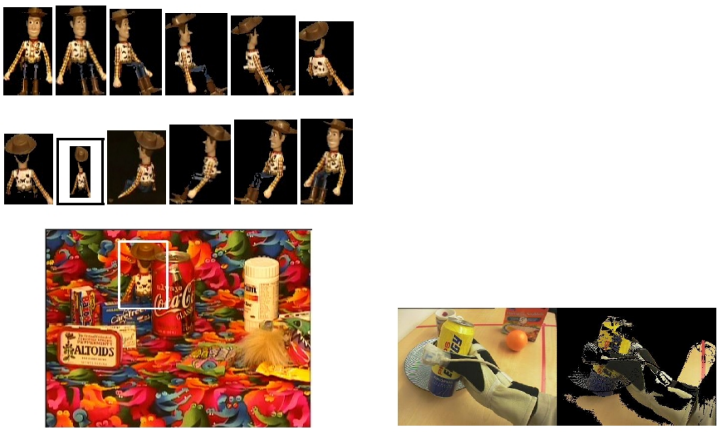

Figure 3.7: CHH applications: (a) model images and typical result of object recognition in spite of background clutter and partial occlusion. The matched model image source [2] (b)Camera view of scene, before and after filtering with the background image, source[3].

3.5 Task Execution

As seen in Figure 3.3 on Page 15, an important part to close the loop in PbD systems is the task execution. This allows us to notice possible errors of the system. In the implementation chapter we explain our simulation execution in order to visualize our grasp mapping errors. However, the implementation of this system on a real robotic platform is the only way for a complete test for our system. Here, three different robotic platform implementations are presented to give an idea of the "next step" for our system execution on real robotic platforms.

An arm/hand system is used in [7] to verify the grasp mapping technique. The arm used is the PUMA 560 arm while the dextrous hand used is the Utah/MIT hand. Both the arm and hand are controlled via Chimera, which is a real-time operating system for sensor-based control developed at Carnegie Mellon University[40]. The joint angles of the arm and the hand are fed from the grasp simulator to the arm/hand system via Chimera.

In [41] and [33] two different robotic platform are presented, with a three fingers Barrett hand attached. They present a complete PbD process implementation. The experimental platform in [41] is a Nomadic Technologies XR4000 with a Puma 560 arm for manipulation Figure 3.8(b) . The robot has sonar sensors, a SICK laser scanner, a wrist mounted force/torque sensor(JR3), and color CCD camera mounted on the Barrett Hand gripper. The palm of the Barrett hand is covered by a VersaPad touch sensor and, on each finger, there is a binocular stereo camera head. The head is equipped with a pair of Sony XC999 cameras, with focal lengths of 18 mm. This system known as Yorick, has four mechanical degrees of freedom; neck pan and tilt, and pan for each camera in relation to the neck.

In [33], the generated robot program is simulated in a graphical environment to find errors, and reject certain actions and movements. After that the program

is executed on a real robotic platform called Albert Figure 3.8(a) . Albert is equipped with a stereo camera head and a 7 DOF light-weight arm. The arm weighs about 35kg and can lift objects of up to 10kg with fully extended modules. A 6 DOF force/torque sensor supplied by the DLR, sensor wrist Munich interconnects the arm with the three finger Barrett hand. The upper body part is mounted on a mobile platform ODETE which was developed at their institute. ODETE is equipped with supersonic sensors and planar SICK laser scanner for self-localization and obstacle avoidance.

They are two similar ways to implement a complete PbD system. The different techniques studied in this project could be easily adapted to these robotics platforms.

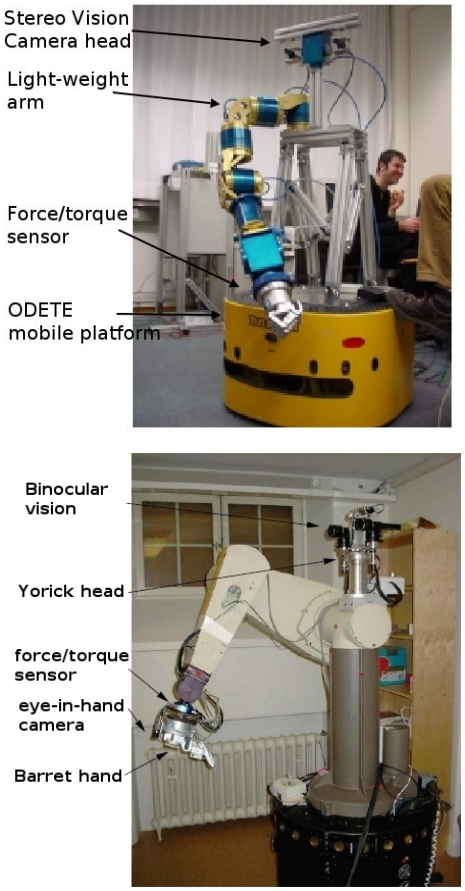

Figure 3.8: Two different robotic platforms: (a) Service robot platform ALBERT (b) Experimental platform on XR4000.

Chapter 4

Implementation

Our main goal is to implement and study different grasp mapping strategies for different objects/manipulators. In order to study different grasp mapping strategies, the system shown in Figure 4.1 was constructed. The system allows us to test several robotic manipulators towards everyday objects. This system consist of three basic PbD, explained in last chapter: data acquisition, grasp mapping modeling and visualization.

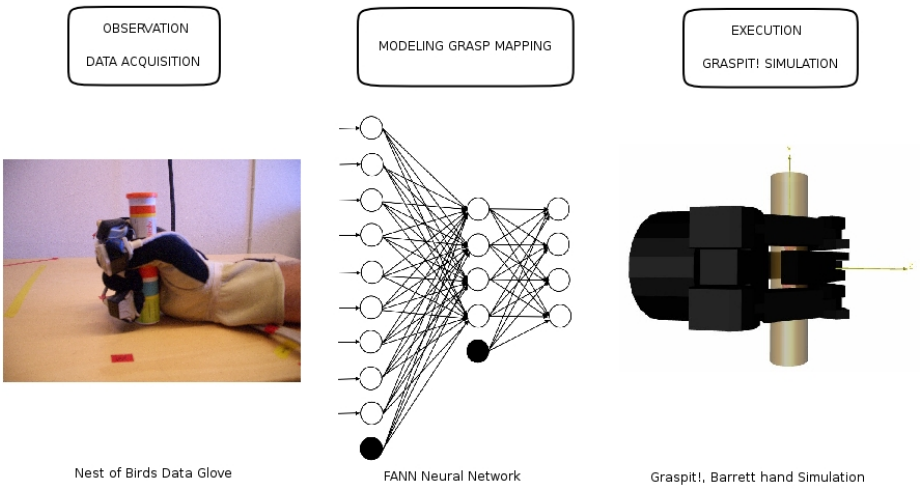

Figure 4.1: Our system structure for intelligent grasp mapping in a PbD framework.

The data is acquired from a human demonstrator wearing a data glove, in our case a magnetic tracking device. The modeling phase contains the main contributions of this project. Here, an ANN is implemented to model the mapping between human and robot space. This network allows us to study different mapping strategies avoiding the need of changing the whole structure for each test. By only providing new training data or changing a few parameters, a new study can be done. In order to do mapping visualization a simulated environment is used. This enables us to repeat the experiment hundreds of times. New

studies with small changes can be done without much effort.

In this chapter more details of the implemented system are given as well as a little background of the components used for this implementation, in order to understand how each part operates, and to understand its role in the complete PbD system implementation.

4.1 Data acquisition

4.1.1 Data Gloves

A Cyber glove or Data glove is an interactive device. Mounting a wired glove on the hand facilitates tactile sensing and fine-motion control in robotics and virtual reality. Data gloves are also one of several types of electromechanical devices used in haptics applications. These gloves are capable of measuring the movements of hand and fingers. A glove is equipped with sensors that sense the movements of the hand and forwards those movements with a computer. The sensors are situated near finger and wrist joints, to detect possible movements on them. Data gloves are commonly used in virtual reality environments where the user sees an image of the data glove and can manipulate the movements of the virtual environment using the glove.

Commercial Data Gloves

Wired gloves are often called "datagloves" or "cybergloves", but these two terms are trademarks. They respectively belong to Sun Microsystems (which acquired the patent portfolio of VPL Research Inc. in February of 1998) and Immersion.

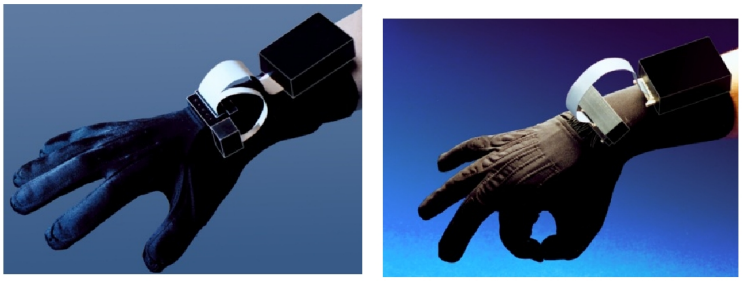

Figure 4.2: Example of two different commercial Data Gloves of 5DT DataGlove Series : (a) 5DT Data Glove 5 (b) 5DT Data Glove 16

More information of this and more commercial Data Gloves in [42], [43], [44]

4.1.2 Magnetic Trackers

Magnetic trackers in general are devices capable of estimating positions and orientations of magnetic sensors. Usually these system consists of an electronic unit, a transmitter generating magnetic fields and several pose measuring sensors. They can give position and orientation by measuring the magnetic fields relative to the transmitter. With this data the the system is capable to track of all sensors and estimate their positions and orientation at 30 Hz.

These sensors can be mounted almost everywhere. For example, on the human body for virtual reality applications or on a moving platform to calculate the position and orientation of it. The disadvantages of these systems are that magnetic fields are often influenced by the environment in such a way that the position and orientation values measured by the sensors contain too much undesirable noise and even sometimes they present extreme points. For this reason we recommended a parallel use with other sensory system, like sensor systems studied in last chapter, to improve the general stability and reliability for the resulting data.

Nest of Birds

The Nest of BirdsFigure 4.3 simultaneously tracks the position and orientation of up to four sensors, providing tracking data, position and orientation of these four sensors in real time. It comes either with a USB or RS-232 interface, and a CD for installation on a windows system. More details and specifications of the product can be seen in [45].

Figure 4.3: Nest of Birds Real-time Motion Tracking

4.1.3 Nest of Birds as a Data Glove

In this project, training sequences are obtained using the Nest of Birds sensor. As seen in the specification NoB is a magnetic tracker that consists of an electronics unit, a transmitter and four pose measuring sensors. The sensors measure transmitter-generated magnetic fields. The electronic unit controls the transmitted signals and directs the sensor measurement. From signals measured by the sensors, Nest of Birds calculates the position and orientation of each sensor.

The motivation for using this sensor is the possibility to mount sensors on different parts of the human body (arm, hand, leg) and easily exchange their configurations. This allows us to learn trajectories and poses for human hand-arm-torso-leg motion and to evaluate different sensor configurations and choose the one that captures the representation space in the best way, an example of this can be seen in [46]. For learning by demonstration frameworks for service robots, and for natural interfaces for human-robot interaction, this approach has much stronger potential compared to dedicated sensors such as the Data Glove. In our project the Nest of Birds individual sensors have been mounted on a glove, as shown in Figure 4.4 .

Figure 4.4: The glove used for measuring human input.

As we can see in Figure 4.4 , the center sensor, mounted on the back side of the glove, serves as a reference sensor. It measures the position and orientation of the hand. The remaining sensors are mounted on the thumb, index finger and little finger, respectively, and provide position measurements. Each of the sensors estimate a 3D-position position in (x, y, z) coordinates and orientation by rotation matrix. More details of this are given in the following Data Modeling section.

Data Modeling

As shown in Figure 4.4 , we have mounted four different sensors on our data glove. We obtain the position of three different fingers in reference to a sensor mounted on the back side of the glove. The fingers can be chosen as it fits the experiment best.

To calculate the relative positions, we use the data structure sent by the NoB. The NoB provides two matrices. First, a position matrix (4x3) which contains three values (x, y, z) for each of the four sensors of NOB. Second, an orientation matrix (4x9) which contains nine values for rotation for each sensor.

The resulting values are calculated according to equation 4.1 , giving a total of nine values. The position of each sensor is represented by $x = (x, y, z)$ and the reference sensor is represented by $x_r = (x_r, y_r, z_r)$. The rotation matrix M is the rotation of the reference sensor, given by the NoB. As seen in equation 4.2 , the features derived from the sensors are both translational and rotational invariant, since the positions are multiplied by the transpose (inverse) of the rotation matrix.

$$p = M^T(x - x_r) \tag{4.1}$$

$$\begin{aligned} px &= (x - x_r)M_{11} + (y - y_r)M_{21} + (z - z_r)M_{31} \\ py &= (x - x_r)M_{12} + (y - y_r)M_{22} + (z - z_r)M_{32} \\ pz &= (x - x_r)M_{13} + (y - y_r)M_{23} + (z - z_r)M_{33} \end{aligned} \tag{4.2}$$

4.2 Modeling using Artificial Neural Networks

The human hand can perform a wide variety of grasps. Because of the difficulties in constructing a robotic gripper equivalent to the human hand, most robotic grippers have much less flexibility. However, the grippers may also have other features, that the human hand cannot compete with. Consequently, the human and robot hand cannot, in general, perform the same types of grasps. A mapping of the grasp type from the human and the robot hand is necessary. As the kinematics and configuration spaces of a human hand and an artificial robot hand are generally different, the fingertip positions of the robot hand cannot correspond exactly to the fingertip positions of the human hand (especially when fingertip grasps are considered).

4.2.1 Motivation for Using Artificial Neural Networks

As mentioned above, a good mapping between the human and the artificial fingers positions is required. The mapping between human hand and robot hand changes in the case of different robotic manipulators. For example, the mappings differ in the case of a three finger robot hand like the Barrett hand, which has only four degrees of freedom, and a five finger robot hand like the Robonaut hand, which has 14 DoFs.

To do an analytical grasp mapping the complex inverse kinematics of the different robotic hands must be solved, for each robot hand. One solution to these differences is to use an Artificial Neural Network (ANN), with which it is possible to adapt the ANN to different robotic hands by only slightly changing some parameters. Knowing the current robotic hand and with knowledge from the robotic hand configuration, it can create the best ANN for the current robotic manipulator.

ANNs allow solving complicated problems with simple and effective solutions. Here grasp mapping is required, for this mapping we use human fingertip positions as input and robotic joint values as output. With ANN it is possible to do specific training between human hand and several robot hands and create the mapping between them. This is easily done with the teacher demonstrating different hand postures to the robot, and using the ANN to train the system to the desired mapped robot pose. The modeling from human grasp to robot grasp is performed during training. These teaching behavior also fits well in the Programming by Demonstration paradigm.

4.2.2 Neural Network Theory

Neural Networks

The human brain is a highly complicated machine capable of solving very complex problems. In order to understand an ANN, we will need to have a basic knowledge of how the internals of how the human brain works. The brain is part of the central nervous system and consists of a very large NN.

The NN is a network consisting of connected neurons. The center of the neuron is called the nucleus. The nucleus is connected to other nucleuses by means of the dendrites and the axon. This connection is called a synaptic connection.

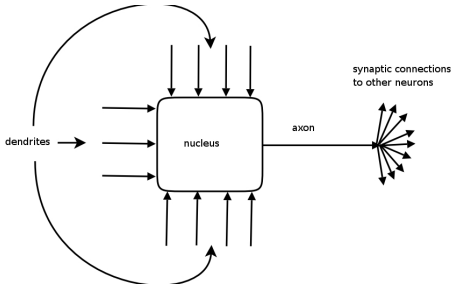

Figure 4.5: Simplified human neuron.

The neuron can fire electric pulses through its synaptic connections, which is received at the dendrites of other neurons. Figure 4.5 shows what a simplified neuron looks like.

When a neuron receives enough electric pulses through its dendrites, it activates and fires a pulse through its axon, which is then received by other neurons. In this way information can propagate through the NN. The synaptic connections change throughout the lifetime of a neuron and the amount of incoming pulses needed to activate a neuron (the threshold) also change. This behavior allows the NN to learn.

The Artificial Neuron

A single artificial neuron can be implemented in many different ways. The general mathematic definition is given in equation 4.3,

$$y(x) = g\left(\sum_{i=0}^{n} w_i x_i\right) \tag{4.3}$$

where x is a neuron with n input dendrites $(x_0 \dots x_n)$ and one output axon $y(x)$ and where $(w_0 \dots w_n)$ are weights determining the importance of each individual input. g is an activation function that weights how powerful the output (if any) should be from the neuron, based on the sum of the input. The output from the activation function is either between 0 and 1, or between -1 and 1, depending on which activation function is used. This is not entirely true, since e.g. the identity function, which is also sometimes used as activation function, does not have these limitations, but most other activation functions uses these limitations. The inputs and the weights are not restricted in the same way and can in principle be between $-\infty$ and $+\infty$, but they have often very small values centered around zero. The artificial neuron is illustrated in equation 4.6 . In the figure of the real neuron Figure 4.5 , the weights are not illustrated, but they are implicitly given by the number of pulses a neuron sends out, the strength of the pulses and how closely connected the neurons are.

As mentioned earlier there are many different activation functions, some of the more commonly used are threshold equation 4.4 , sigmoid Figure 4.5 and hyperbolic tangent equation 4.6 .

$$g(x) = \begin{cases} 1 & \text{if} \quad x + t > 0 \\ 0 & \text{if} \quad x + t \leq 0 \end{cases} \tag{4.4}$$

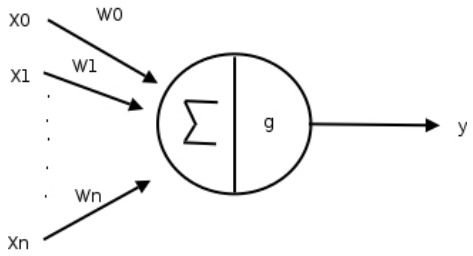

Figure 4.6: An artificial neuron.

$$g(x) = \frac{1}{1 + e^{-2s(x+t)}} \tag{4.5}$$

$$g(x) = tanh(s(x+t)) = \frac{sinh(s(x+t))}{cosh(s(x+t))} = \frac{e^{s(x+t)} - e^{s(x+t)}}{e^{s(x+t)} + e^{s(x+t)}} = \frac{e^{2(s(x+t))} - 1}{e^{2(s(x+t))} + 1} \tag{4.6}$$

Here, t is the value that pushes the center of the activation function away from zero and s is a steepness parameter.

4.2.3 Artificial Neural Networks

It is not possible (at the moment) to make an artificial brain, but it is possible to make simplified artificial neurons and artificial neural networks. These ANNs can be made in many different ways and can try to mimic the brain in many different ways. ANNs are not intelligent, but they are good for recognizing patterns and making simple rules for complex problems. They also have excellent training capabilities which is why they are often used in artificial intelligence research.

Most Artificial Neuron Network (ANN) models are not a very good nor accurate descriptions of biological networks, but should rather be seen as biologically inspired algorithms. An artificial neuron is a very small but essential computational unit, as in the case of biological nervous systems. A single neuron is not very interesting by itself, instead a rather large number of connected neurons are necessary for any substantial computations.

In multilayer feedforward ANNs, which is the most common form of ANNs, the neurons are ordered in layers, starting with an input layer and ending with an output layer. Between these two layers are a number of hidden layers. Connections in these kinds of network only go forward from one layer to the next. Many other kinds of ANNs exists, [47] describes several of these other kinds of ANNs.

Multilayer feedforward ANNs have two different phases: A training phase (sometimes also referred to as the learning phase) and an execution phase. In the training phase the ANN is trained to return a specific output when given a specific input. This is done by continuous training on a set of training data. In the execution phase the ANN returns outputs on the basis of inputs.

A feedforward ANN functions as follows: An input is presented to the input layer. The input is propagated through all the layers (using equation 2.1) until

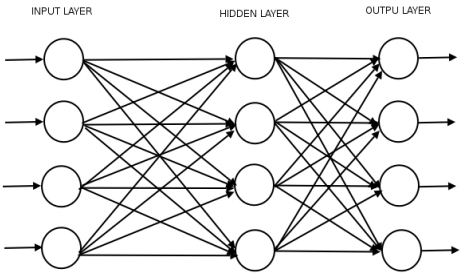

Figure 4.7: A fully connected multilayer feedforward network with one hidden layer.

it reaches the output layer, where the output is returned. In a feedforward ANN an input can easily be propagated through the network and evaluated to produce an output. It is more difficult to compute a clear output from a network where connections are allowed in all directions (like in the brain), since this will create loops.

Figure 4.7 shows a multilayer feedforward ANN where all the neurons in each layer are connected to all the neurons in the next layer. This is called a fully connected network and although ANNs do not need to be fully connected, they often are.

Two different kinds of parameters can be adjusted during the training of an ANN, the weights and the t value in the activation functions. This is impractical and it would be easier if only one of the parameters should be adjusted. To cope with this problem a bias neuron is invented. The bias neuron lies in one layer, is connected to all the neurons in the next layer, but none in the previous layer and it always emits 1. Since the bias neuron emits 1 the weights, connected to the bias neuron, are added directly to the combined sum of the other weights (equation 4.3), just like the t value in the activation functions. A modified equation for the neuron, where the weight for the bias neuron is represented as w_{n+1}, is shown in equation equation 4.7 .

$$y(x) = g\left(w_{n+1}\sum_{i=0}^{n} w_i x_i\right) \tag{4.7}$$

Adding the bias neuron allows us to remove the t value from the activation function, only leaving the weights to be adjusted, when the ANN is being trained. A modified version of the sigmoid function is shown in equation 4.8.

$$g(x) = \frac{1}{1 + e^{-2sx}} \tag{4.8}$$

Is not possible remove the t value without adding a bias neuron, since this would result in a zero output from the sum function if all inputs where zero, regardless of the values of the weights. Some ANN libraries do however remove the t value without adding bias neurons, counting on the subsequent layers to get the right results. An ANN with added bias neurons is shown in Figure 4.8 .

Training an ANN

When training an ANN with a set of input and output data, we wish to adjust the weights in the ANN, to make the ANN give the same outputs as seen in the training data. On the other hand, we do not want to make the ANN too specific, making it give precise results for the training data, but incorrect results for all other data. When this happens, we say that the ANN has been over-fitted.

The training process can be seen as an optimization problem, where we wish to minimize the mean square error of the entire set of training data. This problem can be solved in many different ways, ranging from standard optimization heuristics like simulated annealing, through more special optimization techniques like genetic algorithms to specialized gradient descent algorithms like backpropagation.

The most used algorithm is the backpropagation algorithm (see following section), but this algorithm has some limitations concerning the extent of adjustment to the weights in each iteration. This problem has been solved in more advanced algorithms like RPROP [48] and quickprop [49].

The Backpropagation Algorithm

The backpropagation algorithm works in much the same way as the name suggests: After propagating an input through the network, the error is calculated and the error is propagated back through the network while the weights are adjusted in order to make the error smaller. The explanation is for fully connected ANNs, but the theory is the same for sparse connected ANNs.

Although the purpose is to minimize the mean square error for all the training data, the most efficient way of doing this with the backpropagation algorithm, is to train on data sequentially one input at a time, instead of training on the combined data. However, this means that the order the data is given in is of importance, but it also provides a very efficient way of avoiding getting stuck in a local minima.

A brief explanation of how the backpropation algorithm works: First the input is propagated through the ANN to the output. After this the error of a single neuron and of all the output layer is calculated. The errors and the

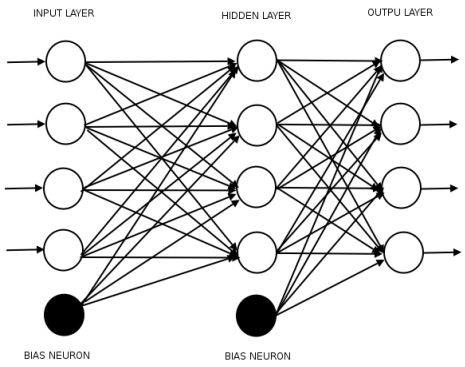

Figure 4.8: A fully connected multilayer feedforward network with one hidden layer and bias neurons.

learning rate parametrer are used for adjusting the weights. The backpropagation algorithm moves on to the next input and adjusts the weights according to the output. This process goes on until a certain stop criteria is reached. The stop criteria is typically determined by measuring the mean square error of the training data while training with the data, and when this error reaches a certain limit, the training is stopped. More advanced stopping criterias involving both training and test data are also used.

This is a short and brief explanation of the backpropagation algorithm without discussion of the mathematics. For a more detailed explanation of the theory behind and the mathematics of this algorithm, see the following references [47],[50].

Neural Networks and Artficial Networks Theory from FAST ARTIFICIAL NEURAL NETWORK LIBRARY (FANN), found in [51].

4.2.4 Fast Artificial Neural Network

The Fast Artificial Neural Network (FANN) library is an ANN library, which can be used with C, C++, PHP, Python, Delphi and Mathematica.

An ANN is normally run in two different modes, a training mode and an execution mode. Although it is possible to do this in the same program, using different programs is recommended.

There are several reasons to why it is usually a good idea to write the training and execution in two different programs, but the most obvious is the fact that a typical ANN system is only trained once, while it is executed many times.

FANN Creation/ Execution

The FANN library is designed to be very easy to use. A feedforward ANN can be created by a simple *fann create standard* function, while other ANNs can be created just as easily.

The library can be used without much knowledge of the internals of ANNs. More advanced users with ANN knowledge can designed customized and specialized networks.

FANN Training

There are several ways of training neural networks and the FANN library supports a number of different approaches. Below two common fundamentally different approaches are presented.

Fixed topology training The size and topology of the ANN is determined in advance and the training alters the weights in order to minimize the difference between the desired output values and the actual output values. This kind of training is supported by fann train on data.

Evolving topology training The training start out with an empty ANN, only consisting of input and output neurons. Hidden neurons and connections is the added during training, in order to reach the same goal as for fixed topology training. This kind of training is supported by FANN Cascade Training.

FANN Training Algorithms

There are four main training algorithms in FANN library.

- **INCREMENTAL**
  Standard backpropagation algorithm, where the weights are updated after each training pattern. This means that the weights are updated many times during a single epoch. For this reason some problems will train very fast with this algorithm, while other more advanced problems will not train very well.
- **BATCH**
  Standard backpropagation algorithm, where the weights are updated after calculating the mean square error for the whole training set. This means that the weights are only updated once during a epoch. For this reason some problems will train slower with this algorithm. But since the mean square error is calculated more correctly than in incremental training, some problems will reach a better solutions with this algorithm.
- **RPROP**
  A more advanced batch training algorithm which achieves good results for many problems. The RPROP training algorithm is adaptive, and does therefore not use the learning rate. Some other parameters can however be set to change the way the RPROP algorithm works, but it is only recommended for users with insight in how the RPROP training algorithm works.
- **QUICKPROP**
  A more advanced batch training algorithm which achieves good results for many problems. The quickprop training algorithm uses the learning rate parameter along with other more advanced parameters, but it is only recommended to change these advanced parameters for users with insight in how the quickprop training algorithm works.

Installing FANN

- Copies of FANN can be obtained from the Source Forge page, located at [52]
- You can currently get FANN as source code (fann-*.tar.bz2), Debian packages (fann-*.deb), or RPM's (fann-*.rpm).
- FANN is available under the terms of the GNU Lesser General Public License.
- FANN bindings to several other programming languages is also available from [52]

4.2.5 FANN Data Modeling

As seen in the NoB data modeling section 4.1.3, nine values are obtained from equation 4.1 to determine the human hand position. Our ANN use these nine variables to represent the input from human hand joint values. These values

have a range approximately between $[12, -12]$, which is the maximum distance from the fingertips to the reference sensor. They range differs for different users.

The Robotic hand joint values are obtained from Graspit!. These values are the output for the network. These values can have a range between $[\pi, -\pi]$, depending on the robot manipulator constrains.

In order to train our ANN, we made tuples of input values representing human hand posture and output values from Graspit!, to link different human hand postures with robotic hand posture.

Our data can be represented with two matrices, an input matrix and an output matrix. The input matrix has same number of rows as number of training postures and nine columns which is the number inputs to the system. The output matrix has the same number of rows and columns as the number of manipulator's DoF. In 4.9 and 4.10 the generalization of these matrices is shown.

$$\mathbf{InputMatrix} = \begin{pmatrix} v_{(1,1)} & \cdots & v_{(1,9)} \\ . & & \\ . & & \\ . & & \\ v_{(numTrain,1)} & \cdots & v_{(numTrain,9)} \end{pmatrix} \tag{4.9}$$

$$\mathbf{OutputMatrix} = \begin{pmatrix} v_{(1,1)} & \cdots & v_{(1,numDoF)} \\ . & & \\ . & & \\ . & & \\ v_{(numTrain,1)} & \cdots & v_{(numTrain,numDoF)} \end{pmatrix} \tag{4.10}$$

In our ANN a symmetric sigmoid function is used in both layers, hidden and output layer. To match the symmetric sigmoid function the input and output values they have to be within $[-1, 1]$ range (FANN also offers the sigmoid function the possibility then the range is $[0, 1]$). A pre-processing of our training data is required, to scale these training values into this range.

To achieve maximum accuracy, our system values are scaled separately by inputs and outputs, but with the same equations. The following steps are done for the input and output matrices.

First, we find the maximum and minimum values of training matrix with equation 4.11 , where max and min are the maximum and minimum values found in the matrix so far, and v is the current process data. This is done for all matrix values.

$$max/min(x) = \begin{cases} max = v & \text{if} \quad (v > max) \\ min = v & \text{if} \quad (v < min) \end{cases} \tag{4.11}$$

After that, we change all the matrix values to the positive side, to make the manipulation easy. This is done with equation 4.15 . Then we change maximum and minimum. With equation 4.13 we find the new max, obviously the new minimum is 0.

$$v = v + abs(\min) \tag{4.12}$$

$$newmax = \max + \min \tag{4.13}$$

Finally, with equation 4.14 we normalize all values to a range between -1 and 1.

$$v = \frac{newmax - v}{newmax} \tag{4.14}$$

In order to visualize the outputs in Graspit! these values have to be rescaled to the original range $(\pi, -\pi)$, so we use the inverse of the same equations, see equation 4.15 and equation 4.16 .

$$v = v - abs(\min) \tag{4.15}$$

$$v = newmax - (v * newmax) \tag{4.16}$$

4.3 Mapping

Nowadays robotic simulation allow researchers, engineers, and students to test control algorithms in a safe environment, recently these systems are able to simulate the dynamics of the mechanism itself and could simulate contacts with other bodies in the environment. That's necessary if the robot's task involves grasping an object, accurate simulation of contact and friction forces is necesited. The following lists we give some reasons to use the simulator environment instead test the different codes and programs in a real environment.

Simulation for hand design

- Mechanical prototypes are costly.
- Spend time and money.
- Evaluate design in simulation.
- Examine range of motion, finger placement, kinematics, link geometry.
- Ensure it can grasp desired objects.
- Change design easily.

Simulation for grasp planning

- Integrated grasp analysis
- Grasp quality, weak point, force optimization
- Visualize grasp from all angles
- Perform many grasps quickly
- Faster than using a real arm and hand
- Build a library of saved grasps
- Recall grasp when object is encountered again

4.3.1 Graspit!

GraspIt! was created to serve as a tool for grasping research. It is a simulator that can accommodate arbitrary hand and robot designs. The user simply specifies the kinematics in a configuration file and provides the link geometry files. It also includes a rapid collision detection and contact determination system that allows a user to interactively manipulate the joints of the hand and create new grasps of a target object. Each grasp is evaluated with numeric quality measures, and visualization methods allow the user to see the weak point of the grasp and create arbitrary 3D projections of the 6D grasp wrench space.

Graspit! a Tool for Grasping Research

- Library of hands and objects.
- Intuitive interface allows many grasps to be tested quickly.
- Visualize grasp wrench space.
- Quality measures evaluate grasp.

GraspIt! also has an automatic grasp planner for the Barrett hand. Given a simplified model of an object constructed from shape primitives, the system can plan a set of candidate grasps, which can then be tested and evaluated. The system can account for the presence of obstacles and the reachability constraints of an attached robot arm. In practice the system can find multiple stable grasps of an object in less than 1 minute. See an example of Graspit! in the Figure 4.9

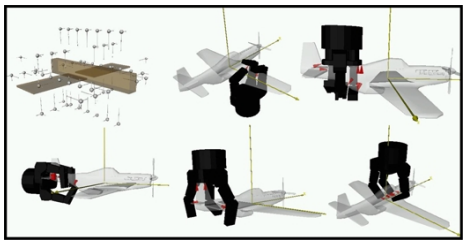

Figure 4.9: The primitive model used for the toy airplane with the generated set of grasps to be tested, and five of the best grasps found sorted in quality order source[4].

Graspit! [4] is a real-time, interactive simulator that allows the user to manipulate a model of an articulated robotic hand, perform grasps with the hand, and visualize stability analysis results on the fly. Graspit has facilities for modeling a robotic arm and workcell to allow the user to examine the reachability constraints for a given grasping task as well. By consistently using accurate geometric models of all elements of the task, we ensure that grasps planned in the simulator can actually be executed in the real world.

GraspIt! makes it possible to import a wide variety of different robot designs. Library of hands currently includes the Barrett Hand, the DLR hand, the NASA Robonaut hand, the Rutgers hand, and a simple parallel jaw gripper, also a puma robot arm. The Graspit! hands are illustrated in Figure 4.10 .

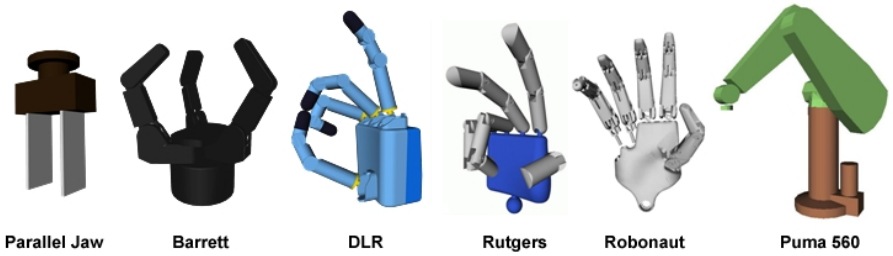

Figure 4.10: The several Robot Hands and arms available in Graspit! simulator

There are two main types of bodies that exist in a GraspIt! simulation world: static bodies (also known as obstacles) and dynamic bodies (such as robot links and objects). Static objects do not participate in the dynamics, but provide collision surfaces for dynamic bodies [53].

Grasping simulator qualities

- Flexible method to specify hand construction.
- Interactive interface.
- Contact determination system.
- Inclusion of material properties.
- Integrated grasp analysis theory into simulation system.
- Grasp Wrench Space GWS, quality measures.
- Visualization techniques.
- Object Dynamics: First implementation of algorithm with large scale and interactive bodies.
- Integration of simulation system into grasping task.
- A useful tool for research in grasping and hand design.
- Currently being used by a number of research groups for design and synthesis: NASA, Sandia, Rutgers, etc.

Chapter 5

Mapping Human Grasps to Robot Grasps

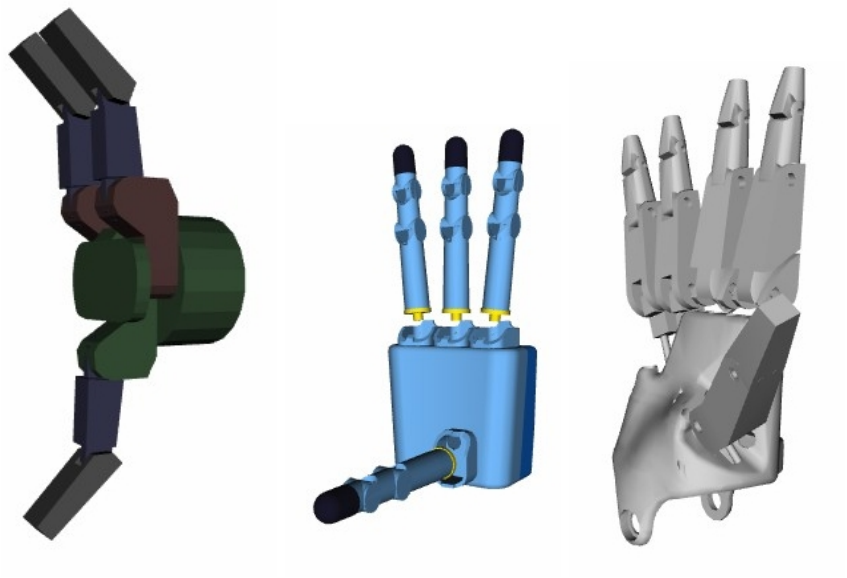

Figure 5.1: Three different robotic manipulators studied in this thesis: (a) Barrett hand (b) DLR hand (c) Robonaut hand.

The system presented in chapter 4 have been implemented to allow us to study and analyze different grasp mapping strategies. The goal of PbD systems is give to robots skills to interact in human natural environments with everyday objects. As examples of everyday objects, we have used two basic shapes, a cube and a cylinder. They can be easily related with a wide list of common objects in the human environment. For the robotic system implementation it is easier to obtain knowledge about basic shapes with different sizes than the full complex model of an object. Also, once the robot has learned grasp mapping to one shape, it can use that knowledge when grasping the same shape with different

size. These are the main motivations to why we focused our work towards these two simples shapes. As an example, any cup or glass is a variation of cylindrical shape, also a lot of common kitchen items like jars, bottles, etc can be related to a basic cylinder shape. The same case occurs with cube shape, there is a huge number of items in our homes that can easily be related to cubic shapes. To see another example, see Figure 5.2 , which shows some of the everyday objects used in training phase.

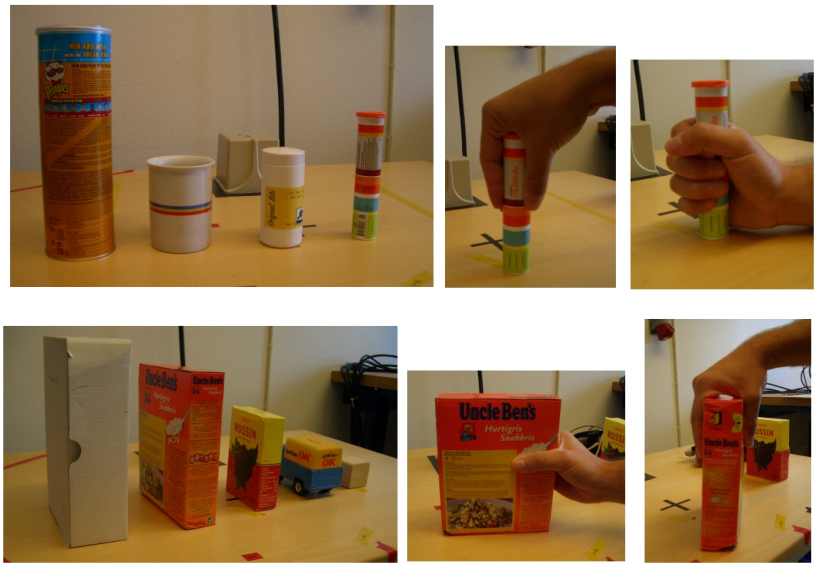

Figure 5.2: Example of the everyday objects used for training: (a) Objects related to cubic shapes and examples of typical grasps trained (b) Objects related with cylinder shapes and examples of typical grasps trained.

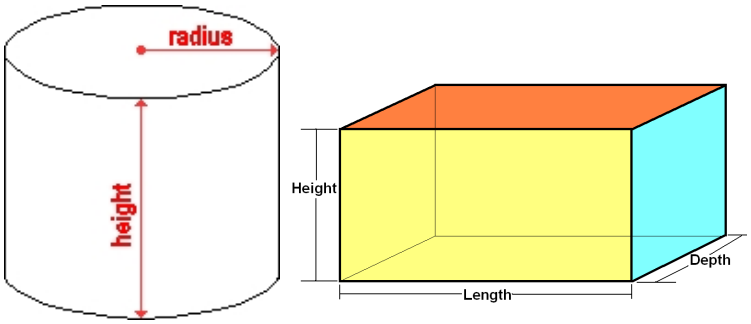

Figure 5.3: Cylinder and cube Shape Cartesian coordinates.

In *Classifying grasps taxonomies*, section 3.3.1, we have seen the hierarchy of possible different human grasps. From this hierarchy and our own human experience we extract three main kinds of grasps for our work: Circular sphere grasp, Power wrap grasp and Precision grasp. They can be seen in Figure 5.2 and Figure 3.6 on Page 20. We think that they are the most commonly used

Object name	Cylinder code	Radius (mm)	Height (mm)
Vitamines	0	15	145
Talc	1	24	105
Cup	2	33	93
Pringels	3	37	230

Table 5.1: Table of size for trained Cylinder objects, with radius and height size in mm.

Object name	Cube code	Depth (mm)	Height (mm)	Length(mm)
Wood cube	0	44	44	44
Toy car	1	62	80	125
Russini	2	33	96	128
Uncle Ben's	3	45	153	190
Carton box	4	70	150	220

Table 5.2: Table of size for trained Cube objects, with Depth, Height and Length size in mm.

grasp for the kind of objects studied.

The task of our system is to do the translation between the grasp performed by human demonstrator to the grasp on a robotic manipulator, adapting it for the robot/object constraints. The Neural Network described in the implementation chapter 4, is used for this purpose. Several network training strategies have been studied, in order to get the best results.

We have studied two different systems. Using the ANN to do direct mapping and using the ANN to do intelligent mapping. Direct mapping is finger position to joint mapping. This results in direct control of the robot movements for the human operator, in other words, direct teleoperation. However, due to sensor noise and limited operator perception, grasping objects using direct teleoperation is still difficult for the operator. In order to get better accuracy when grasping known objects, we made a new system that takes the object's shape and size into consideration. In this way, the system can obtain more precise grasps.

5.1 Direct Mapping for a Robot Hand

The mapping is done directly joint to joint, that means it gives the robotic manipulator the capabilities to do human hand imitation. This system is able to collect fingertip position values from the data glove input, and convert them in robot manipulator joint values.

For direct posture mapping we used an artificial neural network(ANN) created with the FANN library. This network represent the modeling between the user (human hand) and the robot hand configuration spaces. The basic steps involved in this process are i) the sampling of a number of typical human hand postures, ii) matching those directly to robot hand poses and iii) using those for ANN training. The resulting ANN represents the entire mapping surface between these two spaces. An example of a used neural network is a two-layer feedforward network as illustrated in Figure 5.4 .

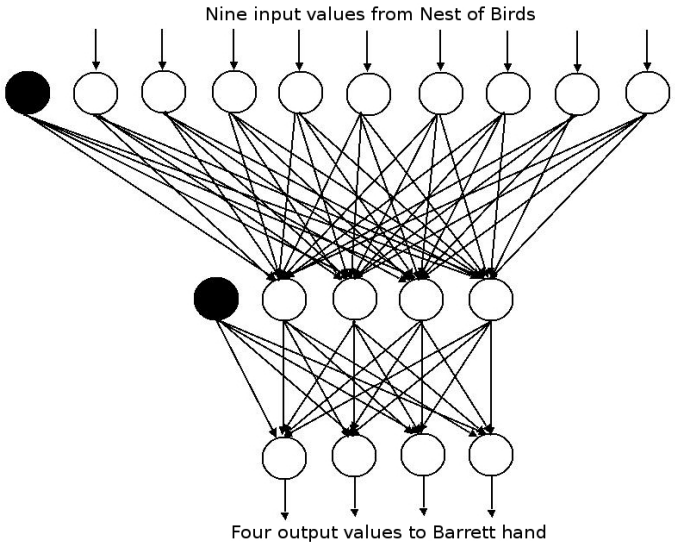

Figure 5.4: Example of a two-layer network used for Barrett hand mapping. There are nine input neurons and four output neurons to fit the number of Barrett DoFs and the same number of hidden neurons. The black circles represents the bias units (always 1).

As seen in previous chapter the FANN library offers a wide list of configurations. Here, the direct mapping use the configuration discussed before. Direct mapping is implemented for three different robotic hands: the Barrett, DLR and Robonaut Hand, shown in Figure 5.1 on Page 40 and presented in following sections. For each manipulator the created ANN has differences, because of the robotic constraints, the number of DoFs, the number of output and hidden neurons and the provided training postures.

5.1.1 Barrett Hand

The Barrett hand shown in Figure 5.1(a) has three fingers and four degrees of freedom (DoFs), one DoF for each finger, the fourth degree is for spread angle. One finger is static only have the possibility to close towards the center, while the movement of two remaining fingers is defined by spread angle and the angle closure for each finger. More information of this robotic manipulator is available in [54].

The kinematic constraints of this hand means that there are many differences between the Barrett and the human hand. This makes controlling the fourth DoF, the spread angle, very difficult. Therefore, this joint was fixed during the first training sequence and only three degrees of freedom were controlled (the angle closure of each finger). However, to demonstrate that the system is capable of controlling all four DOFs, we will present an alternative training scheme in the following section.

Using a Fixed Spread Angle

Despite the big differences between human and Barrett hand, an intuitive joint to joint mapping is possible, if the spread angle is fixed. Note that we have the same number of sensors and DoFs to control. In this sequence the training is done with four simple postures in order to link each finger of the human hand with each finger of Barrett hand, shown in Figure 5.5 .

An Alternative Training Scheme

In Figure 5.6 , is shown some alternative training postures to control the Barrett hand. All DOFs are included, even the spread angle. This example use the most natural way to control this spread angle, from human hand mapping.

In the first training scheme attempting to control the fourth DoF, the spread angle is controlled by moving two sensors, the index and little finger sensors of the NOB data glove. By changing the distance between the fingers, the spread angle is controlled. Increasing this distance will increase the spread angle, and when the distance is reduced the angle is reduced. This mapping shows inefficiency cause the distance change when a finger is moved, for example when grasping an object. This makes control of this DOF difficult and instable.

For these reasons an alternative training scheme is presented, see Figure 5.7 . This setup allows grasping with all robot fingers at once, by using the thumb and the index finger to close the Barrett fingers. Two fingers of robotic hand are controlled with the index finger, and the closure angle of remaining finger is controlled with the thumb. The fourth DOF is controlled by the little finger. As shown in Figure 5.7 , closing the little finger causes the spread angle to close. This type of control is not very intuitive for a human, but may be necessary for certain tasks. Results with this mapping shows more effective grasping than using the intuitive control of our first attempt.

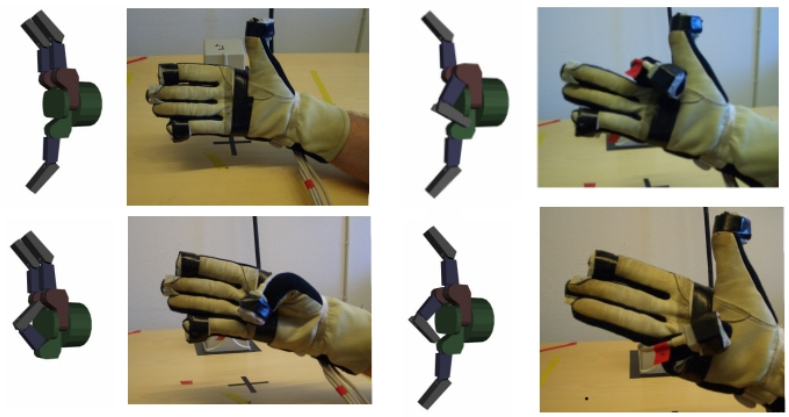

Figure 5.5: Four training postures used for the Barrett hand: Open hand, Thumb closed, Index finger closed and Little finger closed.

Figure 5.6: First alternative training postures to control all DOF of a Barrett hand. The distance between the index and little finger is used to control the spread angle.

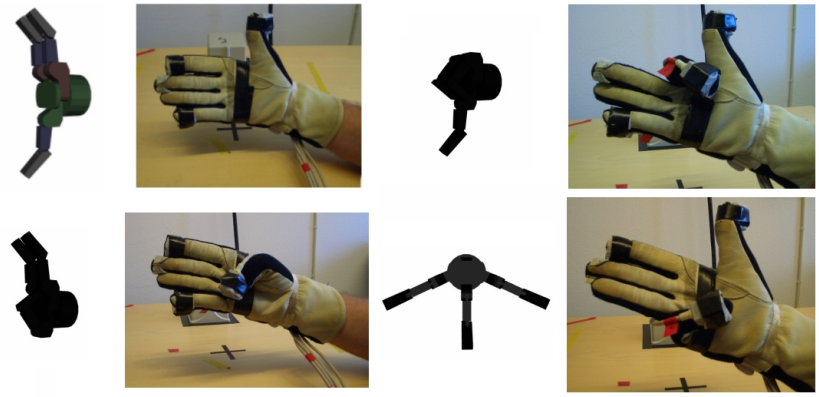

Figure 5.7: Second alternative training postures to control all DOF of a Barrett hand. Two fingers of robotic hand are controlled with the index finger, and the closure angle of remaining finger is controlled with the thumb. The fourth DOF is controlled by the little finger.

5.1.2 DLR Hand

The DLR hand shown in Figure 5.1(b) has four fingers and twelve degrees of freedom: two DOFs for each finger closure and another degree for the spread angle of each finger. This hand has the possibility of closing each finger in two different directions limited for the spread angle. The kinematic constrains make DLR hand different from the Barrett hand, more similar to the human hand. This makes it easier and more intuitive to do joint to joint mapping. However, the spread DOF of each finger is difficult to control using position to joint mapping, and it may cause problems with finger collisions. Therefore, this joint was limited during the first training sequence, and the two remaining DOFs for each finger were directly controlled (the angle closure of each finger). For the DLR hand different training configurations have been tested. Most similar to human motion is controlling the DLR thumb with the human thumb and separate the three remaining fingers into two groups. We decided that the index finger controls the first group and the little finger controls the second group. More information of this robotic manipulator is available in [55]. In Figure 5.8 , we can see the different postures for DLR hand. If more sensors are available, the individual fingers could easily be controlled separately.

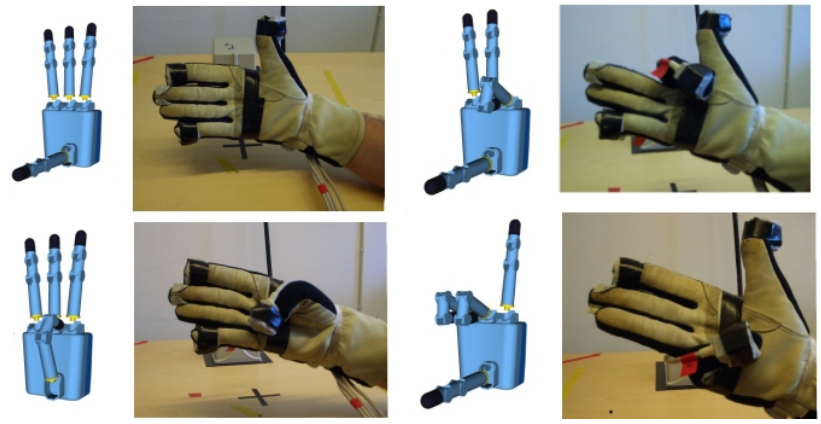

Figure 5.8: Training postures for the DLR hand. Four different postures: open hand, thumb closed, index finger closed and little finger closed that controls two fingers in the DLR hand.

Finger Collisions

A problem in mapping with DLR is that the fingers of the manipulator hand may collide, even when fingers of human hand do not collide, due to spread angle motion. Depending how the mapping is learned this spread angle can cause problems. This represents a problem since if one of the fingers when the hand is closed locks another finger, then the fingers have to be "unfolded" in the correct order, i.e., the finger on top should be opened first. But if the user unfolds his/her grasp in another order, it results in a situation in which the manipulator hand is closed while the human hand is open, then the movements of human hand can not be done for the robot hand until it is unlocked.

To avoid this problem we have limited the spread angle range, with these limits there are less possibilities of collision between fingers.

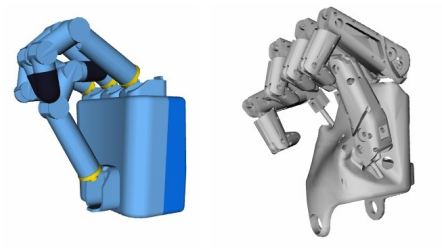

Figure 5.9: Two collision examples for the DLR and Robonaut hand. Here, we can see that thumb finger is locked. To solve the situation the fingers have to be unfolded in correct order.

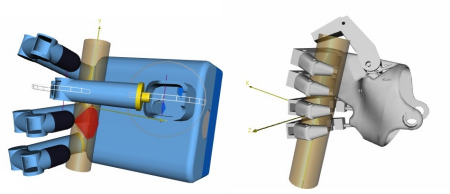

Figure 5.10: Two examples of mapping to avoid colisions for the DLR and Robonaut hand. Here, for cylinder grasp thumb finger is not locked by the others.

5.1.3 Robonaut Hand

The NASA Robonaut Hand shown in Figure 5.1(c) , has a total of fourteen degrees of freedom. There are two DoFs for the wrist, and five fingers with twelve degree of freedom. A dextrous work set allows the hand to maintain a stable grasp while manipulating an object. Training results on this hand are good because control is intuitive and simple, due to the obvious similarities to the human hand. Even in 14-DOF control the ANN has no problems to map the finger positions of the human hand to joint values. The training scheme is similar to the DLR-scheme, where the thumb sensor control the Robonaut thumb and the other four finger can be divided into sections. Note here that the NOB Data Glove does not have enough sensors to cover all fingers. More information on this robotic manipulator is available in [56].

Hence, in our training the little finger or the index finger has to control three fingers simultaneously. Of course, it is possible to demonstrate different behaviors, for example, letting the index and little finger control two fingers each. As said for DLR hand, we are constrained here by the number of sensors. The underlying methodology can easily be extended if more sensors are available.

For this manipulator there is also a big risk of collision, as seen in section 5.1.2. To cope with this, some DOF limitations are given to the direct mapping. The training postures for Robonaut can be seen in Figure 5.11

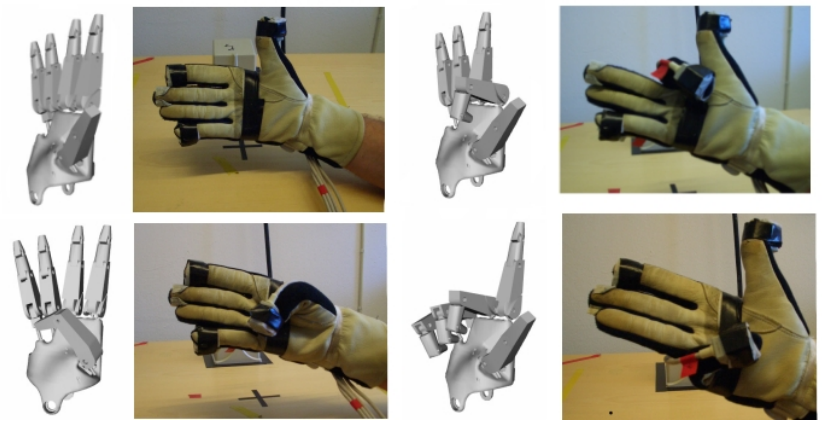

Figure 5.11: Training postures for Robonaut hand. Four different postures are used: open hand, thumb closed, index finger closed and little finger closed.

5.2 ANN training

In order to obtain the best possible mapping with our ANN, some paremetres and options of the FANN library used have to be stipulated and tested in real conditions. In this section present an experimental evaluation of how various parameters affect the result.

5.2.1 Choice of Training Algorithm

As seen in section 4.2.4, the FANN library offers a wide list of options and configurations. Here we have studied how some of the options affect the network. Note our data glove has only three different sensors. For this reason, we chose to start with a network for the Barrett hand, because it has only four DoFs and we can fix one to facilitate the process, see section 5.1.1 for details. We started out with a simple two-layer feedforward neural network as illustrated in Figure 5.4 , with the same number of neurons in the hidden layer than as in the output layer. We trained the network using four basic postures: open hand, thumb closed, index finger closed and little finger closed, shown in Figure 5.5 on Page 44.

Choice of Algorithm

The initial test was done by scaling the data from NoB and Graspit! to symmetric sigmoid range using the equations in section 4.2.4. This test was used for deciding which training algorithm to use, to minimize the error and optimize the training time.

As seen in section 4.2.4, the FANN library offers four different training algorithms: Back-, Rear-, Incremental- and Quick propagation. Fig. Figure 5.12 shows the training execution for all of these algorithms, i.e., how the training evolves for each iteration. The basis of all algorithms is to calculate the mean square error and update the connection weights to minimize this error. Each algorithm has different ways of changing these weights, giving different properties

to the resulting ANN.

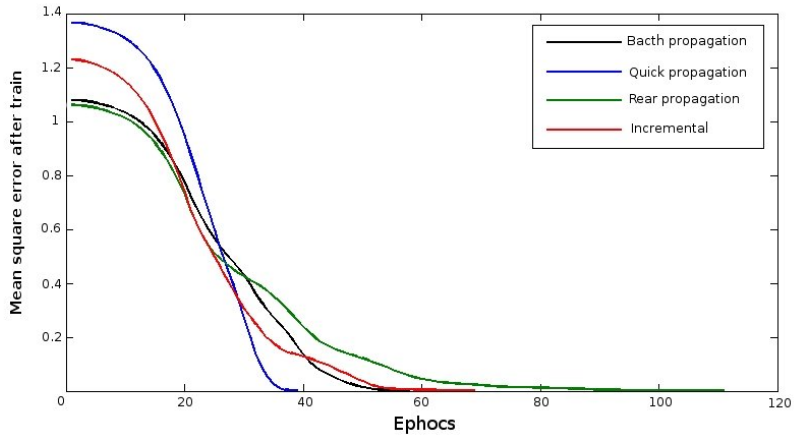

Figure 5.12: ANN training graph for the Barrett hand, for four different FANN algorithms. Black: Batch algorithm, green: Rear propagation, blue: Quick propagation and red: Incremental.

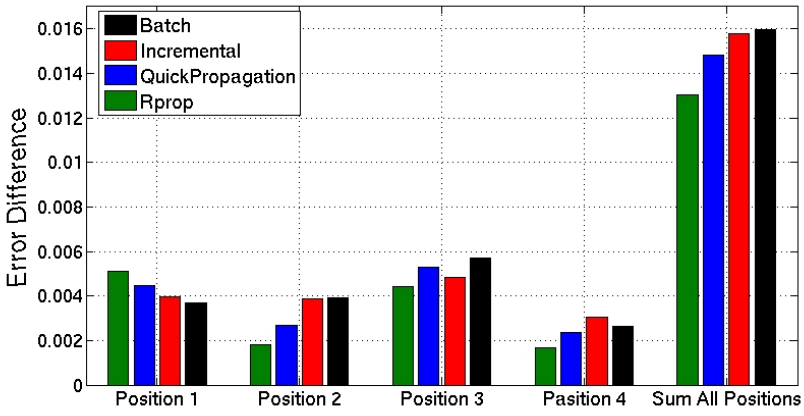

Figure 5.13: Error difference for each trained position and sum of all of them for a Barrett hand ANN, comparison between four different training algorithms. Black: Batch algorithm, green: Rear propagation, blue: Quick propagation and red: Incremental.

We obtain four different networks trained with the same data. Once the network is trained and we measure the training efficiency by calculating the training error, i.e., how well the networks responds to the same input it just has been trained with. This is done by feeding the network with each training position, and then compare the output with each given output. This comparison is done using equation 5.1 , where Difference Error is the squared sum of all DoF differences for this position.

In these graphs we see that the faster training algorithm clearly is Quick propagation (blue color). However, the training error for Quick propagation is not the lowest. Instead, Rear propagation (green color) is giving the least error, although it also comes with the worst training time. Since the training is done offline, the training time is not as important as the training error, so Rear propagation was chosen for this system.

$$PositionError = \sum_{i=1}^{numDoF} (ObtainedDoF_i - ExpectedDoF_i)^2 \tag{5.1}$$

5.2.2 Number of Hidden Neurons

In this section we are interested in how the number of hidden neurons influence the mapping result. We used the same training and test set, and varied the number of hidden neurons. For each test, the difference for every position is calculated according to equation 5.1 and the accumulation of them is used for the graph comparison.

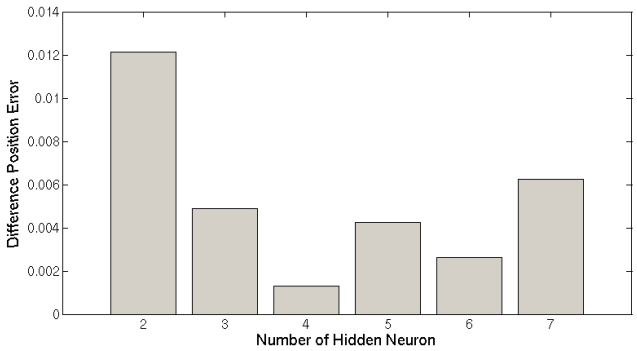

Figure 5.14: Posture error comparison between different number of hidden neurons for Barrett hand training. Note that this graph starts at 2 neurons.

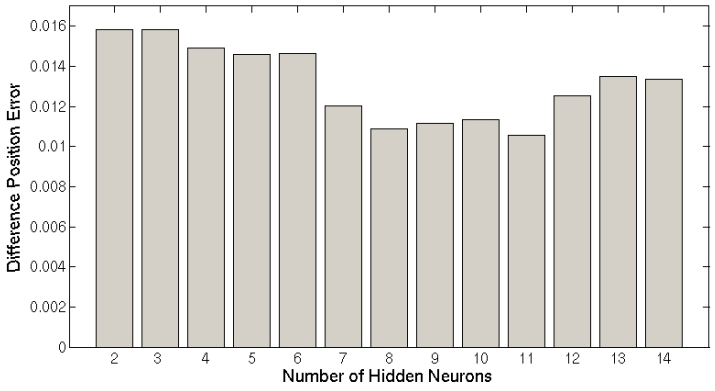

Figure 5.15: Posture error comparison between different number of hidden neurons for DLR hand training. Note that this graph starts at 2 neurons.

For Barrett hand we used the network illustrated in Figure 5.4 . The training algorithm used was Rear propagation. To shown the results in Figure 5.14 on Page 51, we have plotted the accumulated error for the trained positions for the different number of hidden neurons, from 2 to 6 neurons. Here, we see that our assumption was correct; the best number of hidden neurons, four, is same as the number of output neurons for this hand.

For DLR hand we used the network illustrated in Figure 5.4 . However, the number of output neurons must be 12, because the number must be equal to the number DoF to control. Because of this, it is obvious that the number of hidden

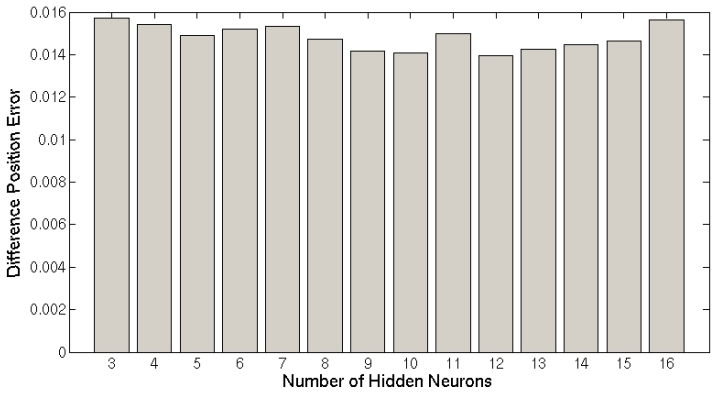

Figure 5.16: Posture error comparison between different number of hidden neurons for Robonaut hand training. Note that this graph starts at 3 neurons.

neurons must increase too. The differences between number of hidden network is discused in the following graph Figure 5.15 . Here is shown the accumulated error, that is obtained from equation 5.1 .

From the graph we see that eight neurons is enough for the direct mapping, and when the number of hidden neurons increase above 12 (number of outputs), the error starts to increase. We chose 8 hiddens neurons for our direct mapping, because despite we could improve error using more neurons the change is not appreciable in execution time. More hidden neurons increases the computational costs and is not necessary for this mapping.

In the Robonaut hand case there are 14 output neurons. For this manipulator the number of hidden neurons has to be larger. In Figure 5.16 , are shown accumulated error difference from 3 to 16 hidden neurons. We can observe that the error is quite stable, but it is smaller between 9 to 13 hidden neurons. For all number of neurons within this interval, direct mapping functioned well. For this reason we have decided to use 9 neurons in order to reduce computational costs.

5.2.3 Number of Training Postures

In this experiment, we evaluated how many training postures were sufficient for effective training of the ANN. Although performed this evaluation for all robot hands, this section only provides results for the DLR hand, since the results for the other robot hands were equivalent. First, we observed that our mapping worked for each hand, but sometimes had problems for unseen hand postures. For example, half-closed finger positions or moving two fingers simultaneously sometimes caused problems. The solution to this problem is to add more training postures. One also has to be careful not to add too many training postures, which may overfit the network.

In this study we test the network with new postures, not shown to the network during training. The result is called a test error and is useful for evaluating how well the network is able to generalize from the training examples. These new postures are shown in Figure 5.17 . They are combination of different

finger movements, which is more difficult for our network to recognize than single finger movements. However, when grasping objects, several fingers always move simultaneously.

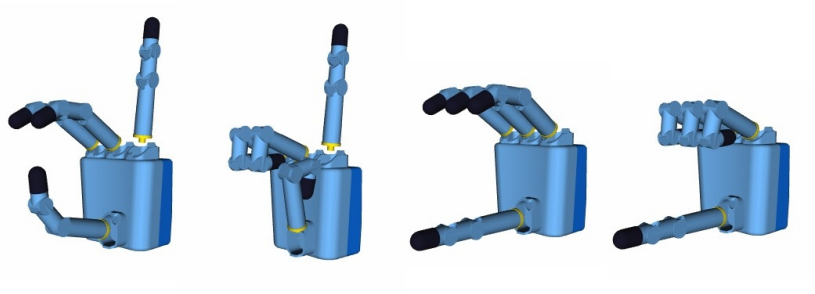

Figure 5.17: Four new test positions for testing the DLR direct mapping network. They are: index/little finger half closed, index/little fingers closed, thumb and little fingers half closed and thumb and little fingers closed.

The error is calculated according to equation 5.2 and equation 5.3 . Hence, we obtain the average error for each posture and we can compare ANNs trained with different number of postures. Results are shown in Figure 5.18 .

$$PositionError = \sum_{i=1}^{numDoF} (ObtainedDoF_i - ExpectedDoF_i)^2 \tag{5.2}$$

$$AverageError = \frac{\sum_{i=1}^{numPositions} PositionError}{numPositions} \tag{5.3}$$

There are two different error graphs, the first, Figure 5.18(a) on Page 54, shows the training error and the second, Figure 5.18(b) on Page 54, shows the test error, the error for non-trained postures seen in Figure 5.17 on Page 53. Here, the new training positions are new intermediate positions for each of the fingers. Thus, for 13 postures, each finger has four training positions: open, a little closed, almost closed, closed. There are three fingers and the final training posture is the open hand.

From the figures, it is easy to observe that the training error increase with number of positions, due to imprecision's and more difficulties for training. Also, training time and mean square error evolution gets worse with more training postures.

On the other hand, the test error decreases with the number of training postures. However, it is possible to see that above 7 postures test error becomes unstable. From Figure 5.18 on Page 54, we can conclude that 7 training postures is the best training configuration for our network, because it is giving adaptability for non-trained positions and we avoid problems like imprecision between training postures and overfitted network.

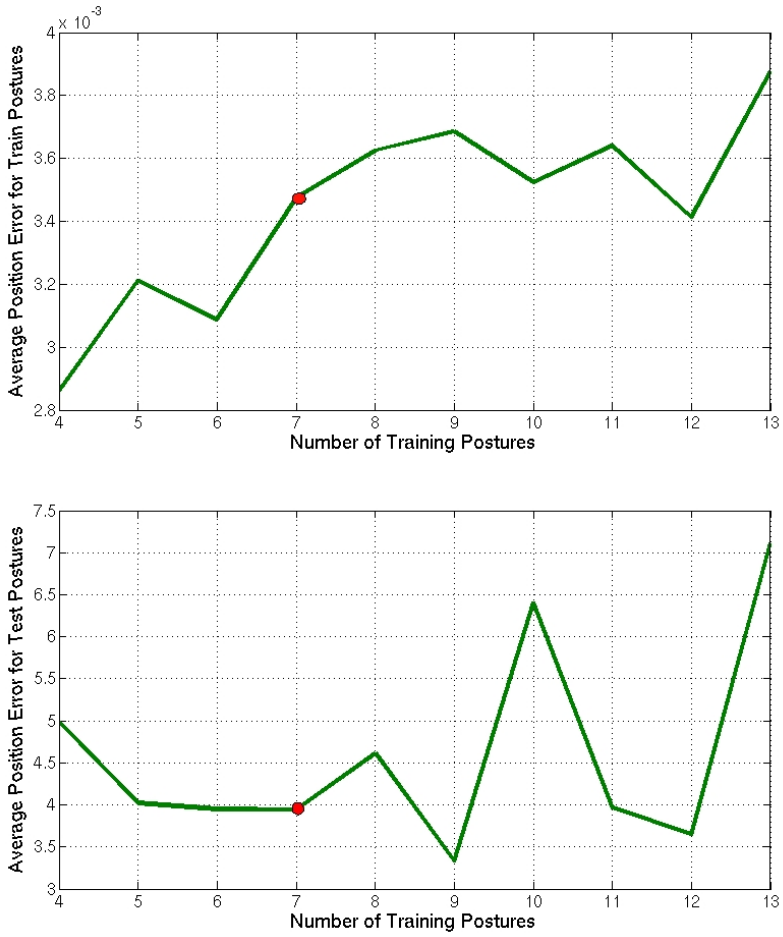

Figure 5.18: Comparison of posture error in DLR network for the number of training postures, Note that starts with four basic postures. (a)Average posture training error. (b) Average posture test error.

5.3 Intelligent mapping based on object shape

In an attempt to improve the grasp mapping when grasping objects, we provide the neural network information about the object that is to be grasped. In order to give this knowledge, we have added a new input neuron for object size in our network. The new network topology is shown in Figure 5.19 .

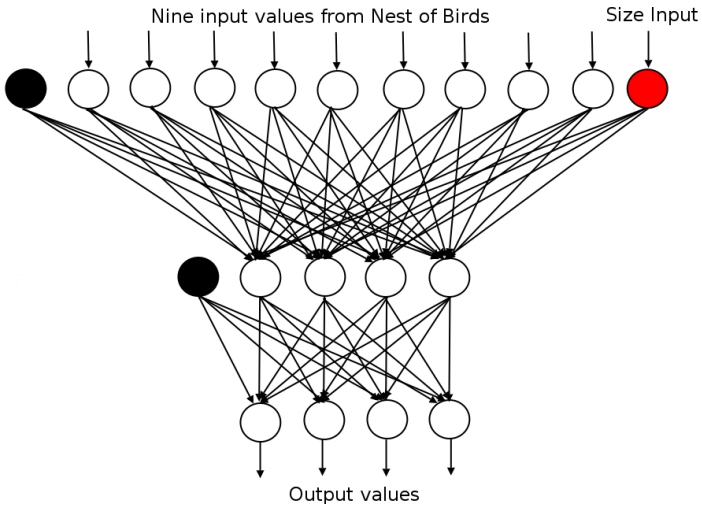

Figure 5.19: New neural network for intelligent mapping implementation, in red the new neuron added for object size.

We have studied this new network with different robotic hands, seen in Figure 5.1 on Page 40, and for different common objects related to two basic shapes, seen in Figure 5.2 . These objects can be releated to real objects with either cylindric or cubic shape.

Chapter 6

Object Grasping Results

6.1 Object Grasping using Direct Mapping

This section shows the results when using direct mapping network to grasp an object. The network was trained for direct mapping, according to the last section, but with new specific training positions for the object grasp task. This test was done for different robotic manipulators and objects. In this section we show the results for the Barrett hand for three cylinders and the Precision grasp. The grasps are shown in Figure 6.1 . For this mapping the network was trained with 10 different positions, seven like direct mapping case and three new positions for each object grasp.

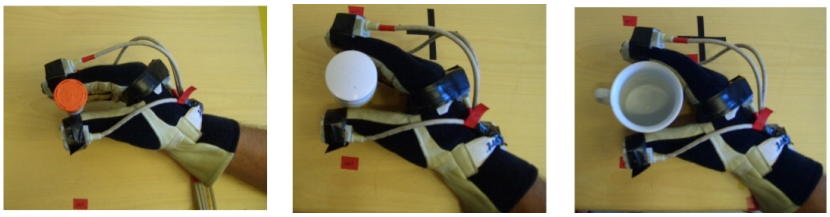

Figure 6.1: Three test grasps towards objects. Precision grasp on three different cylinders.

After training, a test with the same trained positions gives a higher error than the previous test on direct mapping, presented in the last section. However, it is not really comparable since we are not comparing the same postures. It is generally harder to map a complex pose like a grasp, than mapping simple changes for each finger.

In real time simulation on Graspit! this network shows good results. However, still not enough. This network is capable to perform grasps, but not with sufficient precision required for object grasping.

In order to study real capability of this network we test with new data from NoB, for every grasp position. In input acquisition human demonstrator perform the same grasp. However, data acquired is changing lightly, for obvious reason that human actions are not done always strictly equal. The ANN robotic systems need adaptation skills for these changes.

Figure 6.2 shows the network results for this new data. In Gray: same data trained, and in Black: new acquired data. Is obvious the fact that for non trained position error increases, giving more imprecision in real time performance.

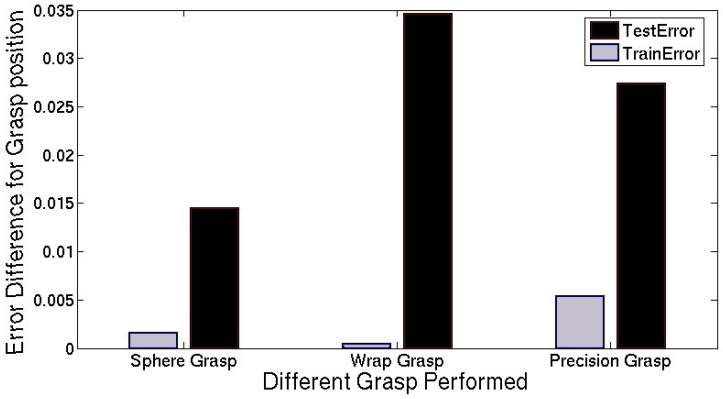

Figure 6.2: Error difference for three different grasps, comparison between trained data and new acquired data. Gray: trained data and Black: not trained data.

Simulating on Graspit! trained data and not trained data, we can see comparison between expected and resulting position. Figure 6.3(a) is network position for data trained, and Figure 6.3(b) is network position for not trained data, in Barrett grasp case. In case of not trained data it is difficult to accomplish the task at hand.

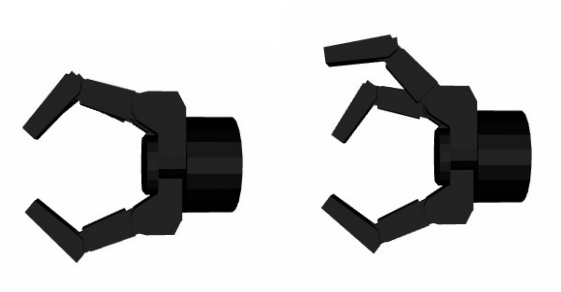

Figure 6.3: Comparison between trained and not trained positions, in grasping with direct mapping network (a)Position with trained data (b) Position with not trained data, from same human position.

We observe that network is capable to perform desired grasp. However, not with desired precision. A new technique to solve the instability of grasp positions is studied in next section.

6.2 Object Grasping Aided by Shape and Size Knowledge

In this section we study a different kind of grasp mapping system: Training new ANNs for grasp mapping targeted to different objects with different shape and varying size. Note that the required number of training data postures increase with number of objects, thereby increasing the comlexity of the system. In this study an ANN for the DLR-hand is used.

6.2.1 Basic Shape: Cylinder

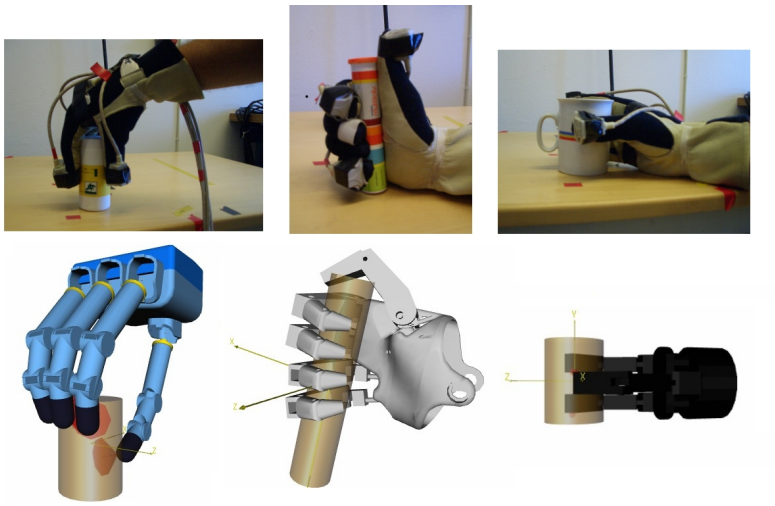

Figure 6.4: Examples of training postures given to the system.

For cylindric shapes, three different grasps have been studied: Circular sphere grasp, Power wrap grasp and Precision wrap grasp, as shown in Figure 6.4 .

As for direct mapping, the 7 basic training postures is used. Three different object sizes are used in training. In addition, three specific grasp positions are used, one for each size. Figure 6.5 shows the comparison between training and test error for the grasp mapping system. Compared to direct mapping, the training error is higher, but more importantly, the test error is much lower. This shows that knowing the object size aids the robot in the grasp mapping process.

In a real system, it is unlikely that the robot will be estimate the size of the object perfectly. To test real system capabilities, we introduced an error to the estimated size of the object. As we can observe in Figure 6.6 , the system recognize the grasp posture. However, it is not able to achieve precice fingertip positions when this error is present.

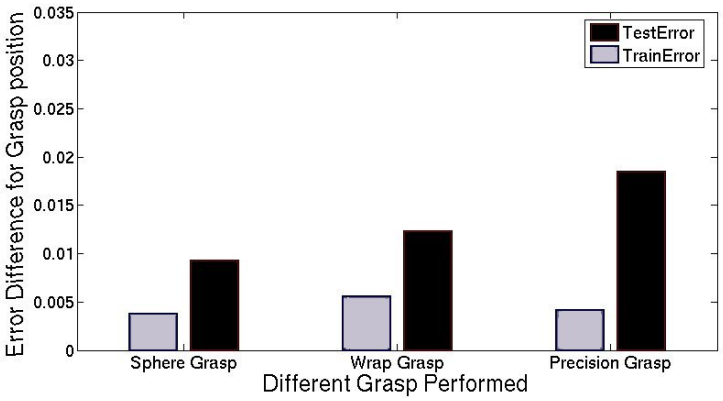

Figure 6.5: Error difference for three different grasps, comparison between training and test error for the same postures.

6.2.2 Basic Shape: Cube

For the analyze of grasp mapping for different cube shapes, we used the Robonaut system. To train and test this system we used four different cubes, shown in Figure 5.2 on Page 41.

Training data is composed for 32 different positions. Seven position like direct mapping for each object size, and four specific grasp positions (as seen above in grasp cylinder training). This training data structure, gives adaptation capabilities for different input size.

After training with this data, the average position error is increasing. As for the cylinder, the training error increases proportional to the number of training positions. For not trained positions it is giving good results, again, better than for the direct mapping network. A comparision between trained and test error is shown in Figure 6.7 .

In this experiment we also test the system for an error in the object size. Partial results are shown in Figure 6.8 . Figure 6.8(a) shows the trained robot grasp posture. Figure 6.8(b) shows grasp posture when the human is grasping a small cube (wood cube see table 5 on Page 42), and the system is told that

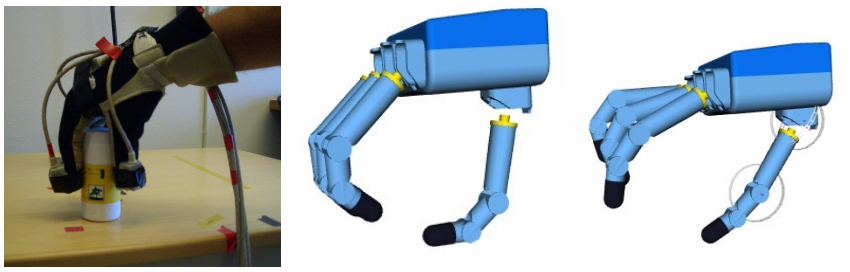

Figure 6.6: Position comparison for cylinder shape and sphere grasp with DLR hand (a) Human position demonstration (b) Network position for trained data (c) Network position for untrained data.

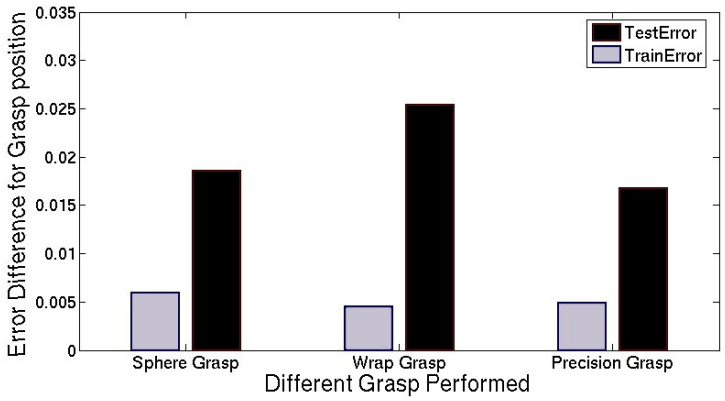

Figure 6.7: Error position difference for three different grasps. Comparison between training and test error for the same human posture.

same small cube is to be grasped. Figure 6.8(c) shows grasp posture when the human is grasping a large cube (carton box see table 5 on Page 42), and the system is still told that a small cube is to be grasped. As seen, this leads to imprecise finger joint values.

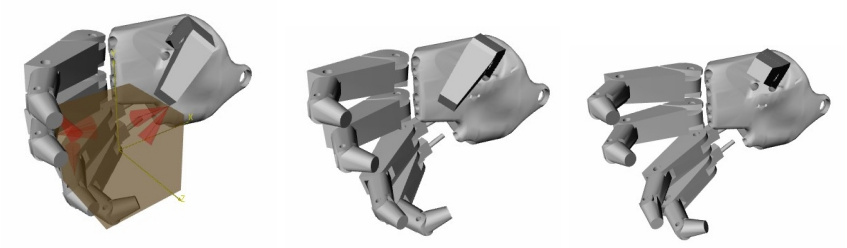

Figure 6.8: Comparison of grasp postures for correct and wrong size in input: (a) Robonaut grasp demonstration (b) Position obtained with correct size in input(wood cube see table 5 on Page 42) (c) Position obtained with wrong size in input (carton box see table 5 on Page 42).

6.3 Object Grasping Comparison

In last sections we have shown results from different grasp mappings. Note that the results are shown for the Barrett hand, with few DOFs, in case of direct mapping, and for the DLR and Robonaut, with many DOFs, for mapping aided by shape and size knowledge. Even though a high number of DOFs would in general give a larger error, we can see that the error is less with the intelligent mapping.

For easier comparison of grasping results, Figure 6.9 shows the results with direct mapping and mapping aided by shape and size knowledge using the Barrett hand and small cube object (see table 5 on Page 42). Here, we can see the differences between the results obtained when the size of object is know for each

grasp type. The main difference is that the error decreses and the final grasp position is more precise. This position allow contact with the target object, and may be helped by a feedback system when contact occurs, from for example touch sensors on the robot fingers. A real robotic platform should be capable to perform this grasp.

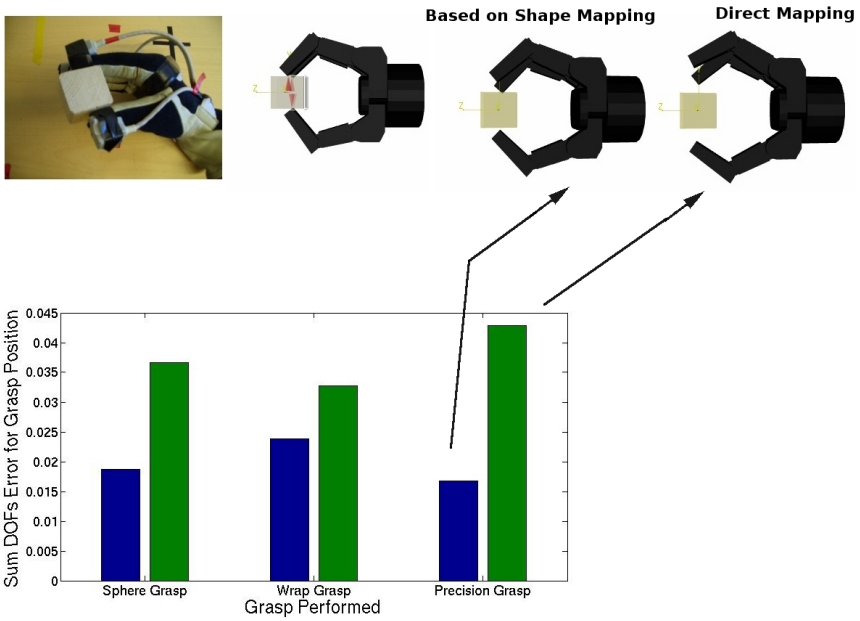

Figure 6.9:

The system has been tested with different robotic hands for every mapping and object. Figure 6.10 and Figure 6.11 show different results for the DLR and Robonaut hand using mapping aided by shape and size knowledge. In these cases two different objects are used: small cylinder and big cube (see table 5 on Page 42). Here, two examples of efficient mapping to allow contact between object and manipulator in a grasp performance are shown.

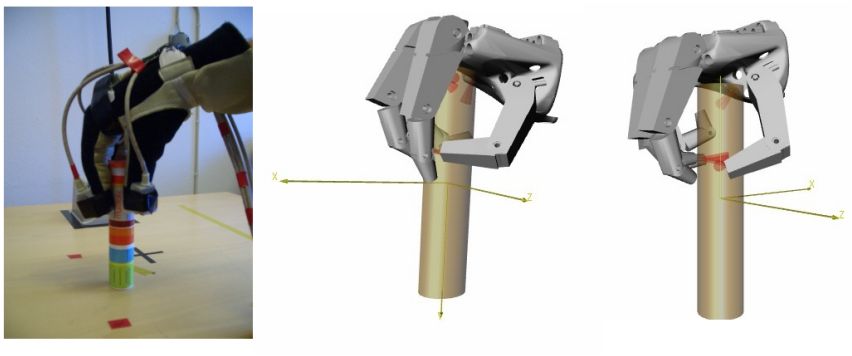

Figure 6.10: Position comparison for small cylinder shape and sphere grasp with Robonaut hand (a) human position demonstration (b) demonstrated grasp position (c) obtained grasp position with mapping aided by shape and size knowledge.

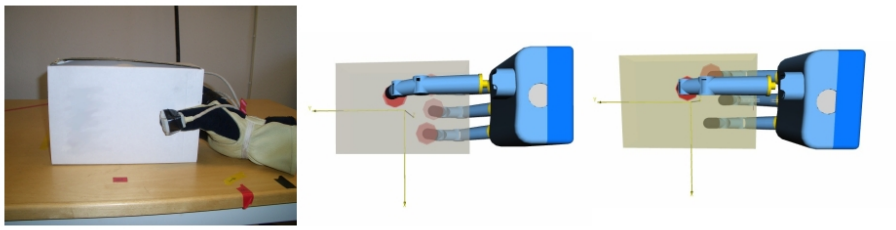

Figure 6.11: Position comparison for big cube and wrap grasp with DLR hand (a) human position demonstration (b) demonstrated grasp position (c) obtained grasp position with mapping aided by shape and size knowledge.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In this thesis, we have studied direct mapping and intelligent mapping for several robotic manipulators, in order to facilitate robot grasping under human control. The experiments have shown that ANN direct mapping between human demonstrator and robotic manipulator gives high movement freedom to the operator, which is able to perform wide list of different postures with the robotic manipulator.

However, for a grasp task we have seen that this mapping is not giving enough precision. Using a new network, the grasp is evaluated and intelligent mapping is performed, adapted for the object's size and shape. This gives better results and more precision in grasp tasks. But, we lose the freedom of the direct mapping, since the mapping is constrained for grasping.

In programming by demonstration and concretely in the field of robotic grasping, it is clear that the robots need a lot of environment information in order to perform complicated tasks. The system can improve the execution if they are able to know *what* they are doing, and *how* they should do it. They can extract this information from human demonstrations if they have a good sensor system and artificial intelligence.

7.2 Future Work

Our system is implemented with a data glove which has limitations in the number of sensors. Our techniques for grasp mapping from human demonstration could be extended in case there are more sensors available. In case of a new data glove which provides finger joint values for the human hand, more accurate postures could be extracted. With such a new configuration system could give better mapping results for robotic manipulators, especially when they have large number of DOFs.

The next step is to test the grasp mappping and teleoperated grasping of real objects, on a real robot platform. We have seen that grasp mapping using

a neural network is working. We have also seen that more information to the network increases performance. Even more information from the human demonstration could be extracted and fed to the network to increase the performance even further.

Bibliography

[1] J. Suomela, From teleoperation to the cognitive human - robot interface, Series A: Research Reports No. 26, Helsinki University of Technology, Automation Technology Laboratory (November 2004).

[2] P. Chang, J. Krumm, Object reconition with color cooccurrence histograms, IEEE Conference on Computer Vision and Pattern Recognition.

[3] S. Ekvall, D. Kragic, Integrating object and grasp recognition for dynamic scene interpretation, In IEEE/RSJ International Conference on Advanced Robotics,ICAR'05, Seattle, USA.

[4] A. T. Miller, Graspit!: A versatile simulator for robotic grasping, Ph.D. thesis, Department of Computer Science, Columbia University, ph.D. Thesis (June 2001).

[5] H. Friedrich, R. Dillmann, O. Rogalla, Interactive robot programming based on human demonstration and advice, Sensor Based Intelligent Robots.

[6] S. Ekvall, D. Kragic, Grasp recognition for programming by demonstration, In IEEE/RSJ International Conference on Robotics and Automation, ICRA'05, Barcelona, Spain.

[7] S. Kang, K. Ikeuchi, Toward automatic robot instruction from perception-mapping human grasps to maniplator grasps, IEEE Transactions on Robotics and Automation 13 (1).

[8] S. Ekvall, D. Kragic, Interactive grasp learning based on human demonstration, In IEEE/RSJ International Conference on Robotics and Automation, ICRA'04, New Orleans, USA.

[9] M. Fischer, P. van der Smagt, G. Hirziger, Lerning techniques in a dataglove based telemanipulation system for the dlr hand, IEEE International Conference on Robotics and Automation 2 (1998) 1603–1608.

[10] M. Cabido-Lopes, J. Santos-Victor, Visual transformations in gesture imitation: what you see is what you do, Proceedings of the 2003 IEEE, International Conference on Robotics and Automation, Taipei, Taiwan.

[11] R. M. Voyles, P. K. Khosla, A multi-agent system for programming robotic agents by human demonstration, american Association for Artificial Intelligence.

[12] Z. Nichol, Y. Liu, P. Suchyta, M. Prokos, A. Goradia, N. Xi, Supermedia enhanced internet-based real-time teleoperation, Tech. rep., Robotics and Automation Laboratory, Michigan State University, East Lansing, MI (2000).

[13] A. Rastogi, P. Milgram, D. Drascic, J. J.Grodski, Virtual telerobotic control, DND Workshop Advanced Technologies in Knowledge Based Systems and Robotics, Ottawa.

[14] I. Elhajj, N. Xi, W. K. Fung, Y.-H. Liu, Y. Hasegawa, T. Fukuda, Supermedia enhanced internet based telerobotics., Proceedings of the IEEE 91 (2003) 396–421.

[15] R. Olivares, C. Zhou, B. Bodenheimer, J. A.Adams, Interface evaluation for mobile robot teleoperation, ACMSE'03.

[16] T. Sheridan, Telerobotics, automation, and human supervisory control, The MIT Press.

[17] S. Tachi, H. Arai, T. Maeda, Development of anthromorphic tele-existence slave robot, Proceedings of the Internacional Conference on Advanced Mechatronics, Tokyo.

[18] T. Fong, C. Thorpe, C. Baur, Collaboration, dialogue, and human-robot interaction, 10th International Symposium on Robotics Research, Lorne, Victoria, Australia, London: Springer-Verlag.

[19] H. Huang, E. Messina, J. Albus, Autonomy level specification for intelligent autonomous vehicles: Interim progress report, Proceedings of the Performance Metrics for Intelligent Systems (PerMI) Workshop, Gaithersburg.

[20] M. Vainio, Intelligence througth interactions - underwater robot society for distributed operations in closed aquatic environment, Tech. rep., Helsinky University of Technology - Automation Technology Laboratory, series A: Research Reports No. 21 (November 1999).

[21] http://www.airforce-technology.com/projects/predator, 10 sep 2006.

[22] http://www.usabilitynet.org/management/bwhat.htm 10 september 2006.

[23] T. Fong, C. Thorpe, Vehicle teleoperation interfaces, Autonomous Robots vol.11 (1).

[24] J. Suomela, A. Halme, Cognitive human machine interface of workpartner robot, Intelligent Autonomous Vehicles Conference (IAV2001), Sapporo, Japan.

[25] R. Pulli, Kaivosautomaatio tnn - koneet ilman kuljettajaa, Automaatio 03 (Automation Days '03), Helsinki, Finland.

[26] D. Spiliotopoulos, I. Androutsopoulos, C. Spyropoulos, Human-robot interaction based on spoken natural language dialogue., Presented at the European Workshop on Service and Humanoid Robots (Servicerob 2001), Santorini, Greece.

[27] T.Fong, F. Conti, S. Grange, C. Baur, Novel interfaces for remote driving: Gesture, haptic and pda, SPIE Telemanipulator and Telepresence Technologies VII, Boston, MA.

[28] O. Rogalla, M. Ehrenmann, R. Zoellner, R. Becher, R. Dillmann, Using gesture and speech control for commanding a robot assistant, Proc. of the 11th IEEE Int. Workshop on Robot and Human interactive Communication, ROMAN2002, Berlin, Germany.

[29] W. Amai, J. Fahrenholtz, C. Leger, Hands-free operation of a small mobile robot, Autonomous Robots vol.11 (1).

[30] R. Krauss, U. Hadar, The role of speech-related arm/hand gestures in word retrieval, r. campbell and l. messing (eds.), gesture, speech, and sign., Tech. rep., Oxford: Oxford University Press (1999).

[31] http://www.speechrecognition.philips.com/products/, 10 september 2006.

[32] T. Williams, One-muscle infant's myoelectric control, Journal of Association of Children's Prosthetic Orthotic Clinics 24 (2).

[33] M. Ehrenmann, O. Rogalla, R. Zllner, R. Dillmann, Teaching service robots complex tasks: Programming by Demonstration for workshop and household environments, Industrial Applications of Informatics and Microsystems, Universitt Karlsuhe (TH), Germany.

[34] L. Petersson, P. Jensfelt, D. Tell, M. Strandberg, D. Kragic, H. Christensen, Systems integration for real-world manipulation tasks., In IEEE International Conference on Robotics and Automation, ICRA 2002 vol. 3 (2002) pages 2500–2505.

[35] M. Cutkosky, On grasp choice, grasp models and the desing of hands for manufacturing tasks, IEEE Transactions on Robotics and Automation (189) 269–279.

[36] H. Friedrich, V. Grossmann, M. Ehrenmann, R. Z. O. Rogalla, R. Dillmann, Towards cognitive elementary operators: grasp classification using neural network classifiers, In IASTED International Conference on Intelligent Systems and Control, Santa Barbara, USA.

[37] T. Wojtara, K. Nonami, H. Shao, R. Yuasa, S. Amano, Y. Nobumoto, Master-slave hand system of different structures, grasp recognition by neural network and grasp mapping, Robotica volume 22 (2004) pages 449–454.

[38] T. Wojtara, K. Nonami, Hand posture detection by neural network and grasp mapping for a master slave hand system, In IEEE/RSJ Intl Conference on Intelligent Robots and Systems(IROS), Sendai, Japan (2004) pages 866–871.

[39] S. Ekvall, F. Hoffmann, D. Kragic, Object recognition and pose estimation for robotic manipulation using color cooccurrence histograms, in IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'03.

[40] D. Stewart, P. Khosla, Chimera 3.1, the real-time prgramming operating system for reconfigurable sensor-based control systems, Tech. rep., Carnegie Mellon University, Pittsburgh (1993).

[41] D. Kragic, S. Ekvall, P. Jensfelt, D. Aarno, Sensor integration and task planning for mobile maniplation, royal Institute of Technology, Stockholm, Sweden.

[42] http://www.5dt.com 10 september 2006.

[43] http://www.vrealities.com/glove.html, 10 september 2006.

[44] http://www.geocities.com/mellott124/dataglove.htm, 10 september 2006.

[45] http://www.ascension-tech.com/products/nestofbirds.php., 15 oct 2006.

[46] I. S. Vicente, Human action recognitin based on linear and non-linear dimensionality reduction using pca and isomap, Master's thesis, Center for Autonomous Systems, Royal Institute of Technology, Stockholm, Sweden (July 2006).

[47] M. H. Hassoun, Fundamentals of artificial neural networks, The MIT Press.

[48] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: The rprop algorithm., In Proc. of the IEEE Intl. Conf. on Neural Networks, San Francisco, CA. (1993) pages 586–591.

[49] S. E. Fahlman, Faster-learning variations on back-propagation: An emperical study. (1988).

[50] J. Hertz, A. Krogh, R. G. Palmer, Introduction to the theory of neural computing., Tech. rep., Addison-Wesley Publishing Company. (1991).

[51] http://fann.sourceforge.net/report/node4.html, 15 october 2006.

[52] http://fann.sf.net, 10 september 2006.

[53] A. T. Miller, H. I. Christensen, Implementation of multi-rigid-body dynamics within a robotic grasping simulator, In Proceedings of the IEEE International Conference on Robotics and Automation (2003) 2262–2268.

[54] http://www.barrett.com/robot/handfram.htm, 20 oct 2006.

[55] http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-398/, 20 oct 2006.

[56] http://robonaut.jsc.nasa.gov/, 20 october 2006.